Relational Parametricity for Computational Effects

Logical Methods in Computer Science
Vol. 5 (3:7) 2009, pp. 1–31
www.lmcs-online.org

Submitted Jul. 16, 2008
Published Aug. 9, 2009

RELATIONAL PARAMETRICITY FOR COMPUTATIONAL EFFECTS

RASMUS EJLERS MØGELBERG [a] AND ALEX SIMPSON [b]

[a] IT University of Copenhagen, Denmark
e-mail address: mogel@itu.dk

[b] LFCS, School of Informatics, University of Edinburgh, Scotland, UK
e-mail address: Alex.Simpson@ed.ac.uk

Abstract. According to Strachey, a polymorphic program is parametric if it applies a uniform algorithm independently of the type instantiations at which it is applied. The notion of relational parametricity, introduced by Reynolds, is one possible mathematical formulation of this idea. Relational parametricity provides a powerful tool for establishing data abstraction properties, proving equivalences of datatypes, and establishing equalities of programs. Such properties have been well studied in a pure functional setting. Many programs, however, exhibit computational effects, and are not accounted for by the standard theory of relational parametricity. In this paper, we develop a foundational framework for extending the notion of relational parametricity to programming languages with effects.

1. Introduction

The theory of *relational parametricity*, proposed by Reynolds [32], provides a powerful framework for establishing properties of polymorphic programs and their types. Such properties include the "theorems for free" of Wadler [41], universal properties for datatype encodings, and representation independence properties for abstract datatypes. These results are well established, see e.g. [29], for the pure Girard/Reynolds second-order λ-calculus (a.k.a. system F) which provides a concise yet remarkably powerful calculus of typed total functions.

The generalisation of relational parametricity to richer calculi can be problematic. Even the addition of recursion (hence nontermination) causes difficulties, since the fixed-point property of recursion is incompatible with certain consequences of relational parametricity as usually formulated.[1] This issue led Plotkin [28] to propose using second-order linear type theory as a framework for combining parametricity and recursion, an idea which has since

1998 ACM Subject Classification: F.3.2, D.3.3.
Key words and phrases: Relational parametricity, computational effects, monads, intuitionistic set theory.
[a] Research supported by EPSRC and the Danish Agency for Science, Technology and Innovation.
[b] Research supported by an EPSRC Advanced Research Fellowship.

[1] Relational parametricity implies types form a cartesian closed category with finite sums, and any such category with fixed points is trivial.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-5 (3:7) 2009

been developed in an operational setting in [3] and in a denotational setting in [4]. One of the many good properties of the resulting theory of linear parametricity is that it supports a rich collection of polymorphic datatype encodings with the desired universal properties following from relational parametricity.

The addition of recursion is just one possible extension of second-order λ-calculus. In [9], M. Hasegawa develops a syntactic account of relational parametricity for an orthogonal extension obtained by adding control operators (such an extension was first introduced by Parigot [24] for proof-theoretic purposes). An intriguing fact he observes is that, even though the technical frameworks for the two approaches are quite different, there are striking analogies between his "focal" parametricity and Plotkin's linear parametricity. Accordingly, Hasegawa poses the question of whether it is possible to find a unifying framework for relational parametricity that includes both his work and Plotkin's linear parametricity as special cases.

In this paper we provide a general theory of relational parametricity for computational effects, which answers Hasegawa's question in the affirmative. Not only does our approach generalise both Plotkin's and Hasegawa's, but it also applies across the full range of computational effects (e.g., nondeterminism, probabilistic choice, input/output, side effects, exceptions, etc.).

We build on the work of Moggi [22, 23], who proposed incorporating effects into type theory by adding a new type constructor for typing "computations" rather than values. For every type B, one has a new type $!\mathsf{B}$ (our non-standard notation is justified in Section 5) whose elements represent computations that (potentially) return values in B, and which (possibly) perform effects along the way. Semantically, $!$ is interpreted using a *computational monad* that encapsulates the relevant kinds of effect.

In order to obtain an account of relational parametricity for monads, one needs to solve a problem. Basic to relational parametricity is the idea of treating types as relations. Polymorphic functions are required to preserve derived relations under all possible instantiations of relations to type variables. To extend this to computational effects it is necessary to determine how the operation $!$ determines a relation $!R \subseteq !\mathsf{A} \times !\mathsf{B}$ from any relation $R \subseteq \mathsf{A} \times \mathsf{B}$. That is, one needs a "relational lifting" of the $!$ operation. The literature contains two approaches to defining such a relational lifting for $!$ [8, 14] (although neither is presented in the context of polymorphism). Rather than choosing between these approaches, we instead side-step the issue in a surprising way: we show that, given the right choice of underlying type theory, $!$ is polymorphically definable in terms of more basic primitives whose relational interpretations are immediately apparent.

Our type theory, which we call PE, is presented in Section 2. It is closely related to Levy's system of *call by push-value (CBPV)* [15], which subsumes call-by-name and call-by-value calculi with effects. Levy, following the lead of Filinski [5], emphasises the importance of having two general classes of types: *value types*, which classify "values", and *computation types*, which classify "computations". The intuitive difference between the two is that "a value *is*" and "a computation *does*". Technically, this intuition is supported by the vast range of semantic and operational interpretations of the framework, see [15].

With general computation types at hand, one can give the $!$ constructor the following polymorphic definition:

$$!\mathsf{B} \;=_{\mathrm{def}}\; \forall \underline{X}.\, (\mathsf{B} \to \underline{X}) \to \underline{X} \qquad (\underline{X} \text{ not free in } \mathsf{B}), \tag{1.1}$$

where importantly the type variable $\underline{X}$ ranges over computation types only. As we shall see, the type constructors used in the definition all have natural relational interpretations, and hence the defined ! operation inherits an induced relational lifting.

In order to reason about parametricity in PE, we build a relationally parametric model of our calculus. Even in the case of ordinary second-order λ-calculus, the construction of parametric models is a nontrivial task. In our case, the interaction between value and computation types contributes significant additional complexity. To keep things as simple as possible, we work with a set-theoretic model, exploiting the fact that it is consistent to do so if one keeps to intuitionistic reasoning. The details are presented in Sections 3 and 4. As a first application of the model, we prove in Section 5 that the ! operator, as defined by (1.1) above, does indeed enjoy its expected universal property (Theorem 5.2).

In Section 7, we consider how to specialise the generic calculus PE to specific effects of interest. One useful form of specialisation recurs in many examples. It is common for effects to have associated operations that trigger and/or react to "effectful" behaviour. Typically, one would like to give an n-ary such operation the polymorphic type:

$$\forall X.\ (!X)^n \to !X\ . \tag{1.2}$$

For example, a binary nondeterministic choice operation forms a computation by choosing between two possible continuation computations. Also, the "handle" operation for an exception e, can be viewed as a binary operation where $\mathrm{handle}^e(p, q)$ behaves like p unless p raises exception e, in which case q is executed. Since such operations are computed in a type-independent way, they are "parametric" in the informal sense of Strachey. We show that such operations are also parametric according to our theory of relational parametricity. This involves two technical developments, each of interest in its own right. The first relates to recent work by Plotkin and Power [31], in which they observe that many operations on effects are "algebraic operations" in the sense of universal algebra. As Theorem 7.1, we obtain that n-ary algebraic operations are in one-to-one correspondence with (parametric) elements of type:

$$\forall \underline{X}.\ \underline{X}^n \to \underline{X}\ , \tag{1.3}$$

where again $\underline{X}$ ranges over computation types. Thus algebraic operations can be incorporated within PE as constants of the above type (which is more informative than (1.2), since monadic types $!\mathsf{B}$ are always computation types).

Not all useful operations on effects arise as algebraic operations; e.g., exception handling is a counterexample. However, exception handling can be added to PE using a different strengthening of (1.2) for its type:

$$\forall X.\ (!X)^2 \multimap !X\ . \tag{1.4}$$

This is indeed a strengthening of (1.2) because the lollipop can be understood as restricting the full function space to a subclass of "linear" (in a sense to be explained in the sequel) functions. This correctness of the above typing is again based on a general result (Theorem 7.2) which characterises the (parametric) elements of the above type in terms of a naturality condition.

In Section 8, we outline the relationship between PE and other approaches to parametricity and effects. Plotkin's linear parametricity arises as a specialisation of PE valid in the special case of "commutative" monads. We also briefly discuss how Hasegawa's account of parametricity and control arises as a specialisation of PE. The details for this appear in a companion paper [20]. Finally, in Section 9, we discuss how the theory established in

$$\frac{}{\Gamma,\, x\colon \mathsf{B} \mid - \vdash x\colon \mathsf{B}} \qquad \frac{\Gamma,\, x\colon \mathsf{B} \mid \Delta \vdash t\colon \mathsf{C}}{\Gamma \mid \Delta \vdash \lambda x\colon \mathsf{B}.\, t\colon \mathsf{B} \to \mathsf{C}} \qquad \frac{\Gamma \mid \Delta \vdash s\colon \mathsf{B} \to \mathsf{C} \quad \Gamma \mid - \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash s(t)\colon \mathsf{C}}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \Lambda X.\, t\colon \forall X.\, \mathsf{B}}\ X \notin \mathrm{ftv}(\Gamma, \Delta) \qquad \frac{\Gamma \mid \Delta \vdash t\colon \forall X.\, \mathsf{B}}{\Gamma \mid \Delta \vdash t(\mathsf{A})\colon \mathsf{B}[\mathsf{A}/X]}$$

$$\frac{}{\Gamma \mid x\colon \underline{\mathsf{A}} \vdash x\colon \underline{\mathsf{A}}} \qquad \frac{\Gamma \mid x\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}}{\Gamma \mid - \vdash \lambda^{\circ} x\colon \underline{\mathsf{A}}.\, t\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid - \vdash s\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \quad \Gamma \mid \Delta \vdash t\colon \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash s(t)\colon \underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \Lambda \underline{X}.\, t\colon \forall \underline{X}.\, \mathsf{B}}\ \underline{X} \notin \mathrm{ftv}(\Gamma, \Delta) \qquad \frac{\Gamma \mid \Delta \vdash t\colon \forall \underline{X}.\, \mathsf{B}}{\Gamma \mid \Delta \vdash t(\underline{\mathsf{A}})\colon \mathsf{B}[\underline{\mathsf{A}}/\underline{X}]}$$

Figure 1: Typing rules.

this paper might be applied to derive operational properties of polymorphic languages with effects.

2. A polymorphic calculus

We start by defining the type theory PE for polymorphism and effects. As discussed in the introduction, following [15], PE contains both *value types* $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ and *computation types* $\underline{\mathsf{A}}, \underline{\mathsf{B}}, \underline{\mathsf{C}}, \ldots$. A central feature of our type theory is that we allow polymorphic type quantification over both value types and computation types. Accordingly, we use $X, Y, Z, \ldots$ to range over a countable set of value-type variables, and $\underline{X}, \underline{Y}, \underline{Z}, \ldots$ to range over a disjoint countable set of computation-type variables. Value types and computation types are then mutually defined by:

$$\begin{aligned} \mathsf{A} &::= X \mid \mathsf{B} \to \mathsf{C} \mid \forall X.\, \mathsf{B} \mid \underline{X} \mid \forall \underline{X}.\, \mathsf{B} \mid \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \\ \underline{\mathsf{A}} &::= \mathsf{B} \to \underline{\mathsf{A}} \mid \forall X.\, \underline{\mathsf{A}} \mid \underline{X} \mid \forall \underline{X}.\, \underline{\mathsf{A}} \end{aligned}$$

Note that the computation types form a subcollection of the value types. The intuition here is that any (active) computation has a corresponding (static) value, its "thunk". In contrast to [15], we make this passage from computations to values syntactically invisible.

For semantic intuition, one can think of value types as representing sets, and of computation types as representing Eilenberg-Moore algebras for some computational monad on sets. Then $\mathsf{B} \to \mathsf{C}$ is the set of all functions. The special case $\mathsf{B} \to \underline{\mathsf{A}}$ is a computation type because algebras are closed under powers, with the algebra structure defined pointwise. The type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ represents the set of all algebra homomorphisms from $\underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$. In general, there is no natural algebra structure on this set, hence the type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ is not a computation type. Finally $\forall X.\, \mathsf{B}$ and $\forall \underline{X}.\, \mathsf{B}$ are polymorphic types, with the polymorphism ranging over value types and computation types respectively. In either case, when B is a computation type, the polymorphic type is again a computation type. This is justified by Proposition 4.1 below.

Our types, which are based on function spaces and polymorphism, are not directly comparable with Levy's [15], which include sums and products. Nonetheless, we shall see in

Section 8 that we can encode Levy's calculus within ours. Given this, our calculus extends Levy's with polymorphic types (cf. [15, §12.4]) and linear function types. The latter have a particularly nice explanation in terms of Levy's stack-based operational framework, within which a value of type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ can be understood as a stack turning a computation of type $\underline{\mathsf{A}}$ into a computation of type $\underline{\mathsf{B}}$, cf. [16]. In our system, linear function types will be used crucially in the computation-type encodings of Section 8.

Having computation types as special value types allows us to base our type system on a single judgement form:

$$\Gamma \mid \Delta \vdash t \colon \mathsf{B} \ ,$$

where Γ and Δ are disjoint contexts of variable typings subject to the following conditions: either (i) Δ is empty, or (ii) B is a computation type and Δ has the form $x \colon \underline{\mathsf{A}}$, where $\underline{\mathsf{A}}$ is also a computation type. Thus the context Δ, which, following [6, 7], we call the *stoup* of the typing judgement, contains at most one typing assertion. When we want to be explicit about which of (i) or (ii) applies, we write:

$$\begin{array}{ll} \text{(i)} & \Gamma \mid - \vdash t \colon \mathsf{B} \\ \text{(ii)} & \Gamma \mid x \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}} \ . \end{array}$$

In the first case, the intuitive interpretation of t is as an arbitrary function from the product of all types in Γ to the type B. In the second case, the interpretation of t is as a function from $\Gamma \times \underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$ that is an algebra homomorphism in its right-hand argument (i.e., for every fixed set of values for the Γ variables, the induced function from $\underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$ is a homomorphism). From this interpretation, one sees why the stoup is restricted to computation types, and also why, when the stoup is nonempty, the result type is required to be a computation type.

The type system is presented in Figure 1. The side conditions refer to the set $\mathrm{ftv}(\Gamma)$ of free type variables in a context Γ, which is defined in the obvious way. Of course, the type rules are restricted to apply only when the premises satisfy the conditions on judgements imposed above. In such cases, the rule conclusions also satisfy these conditions.

The following simple lemmata state basic properties of the type system.

Lemma 2.1 (Unicity of types)**.** *For any Γ, Δ, t there is at most one type B such that $\Gamma \mid \Delta \vdash t \colon \mathsf{B}$.*

Lemma 2.2 (Substitution)**.**
(1) *If $\Gamma, x \colon \mathsf{A} \mid \Delta \vdash t \colon \mathsf{B}$ and $\Gamma \mid - \vdash s \colon \mathsf{A}$ then $\Gamma \mid \Delta \vdash t[s/x] \colon \mathsf{B}$.*
(2) *If $\Gamma \mid x \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$ and $\Gamma \mid \Delta \vdash s \colon \underline{\mathsf{A}}$ then $\Gamma \mid \Delta \vdash t[s/x] \colon \underline{\mathsf{B}}$.*

Proof. Both statements are proved by induction over the depth of the typing derivation for t. For example, consider the second statement in the case of $t = u\,u'$, where $\Gamma \mid x \colon \underline{\mathsf{A}} \vdash u \colon \mathsf{C} \to \underline{\mathsf{B}}$ and $\Gamma \mid - \vdash u' \colon \mathsf{C}$. In this case $(u\,u')[s/x] = u[s/x]\,u'$ and by induction hypothesis $\Gamma \mid \Delta \vdash u[s/x] \colon \mathsf{C} \to \underline{\mathsf{B}}$, so $\Gamma \mid \Delta \vdash u[s/x]\,u' \colon \underline{\mathsf{B}}$. □

It is immediate that the type system for value types extends the standard second-order λ-calculus of Girard and Reynolds. Indeed, the typing rules for the relevant types (X, $\mathsf{B} \to \mathsf{C}$ and $\forall X.\, \mathsf{B}$), when restricted to the case with empty stoup, are just the usual ones. It is well-known that the second-order λ-calculus is powerful enough to encode many type constructors including products, sums, inductive and coinductive types. We include those definitions we shall need later in Figure 2. These encodings are all standard apart from the last one which is existential quantification over computation types. The introduction and elimination constructs for the definable value types are encoded in most cases as in the

$$
\begin{array}{rll}
1 & =_{\mathrm{def}} \forall X.\, X \to X & \\
\mathsf{A} \times \mathsf{B} & =_{\mathrm{def}} \forall X.\, (\mathsf{A} \to \mathsf{B} \to X) \to X & (X \notin \mathrm{ftv}(\mathsf{A},\mathsf{B})) \\
0 & =_{\mathrm{def}} \forall X.\, X & \\
\mathsf{A} + \mathsf{B} & =_{\mathrm{def}} \forall X.\, (\mathsf{A} \to X) \to (\mathsf{B} \to X) \to X & (X \notin \mathrm{ftv}(\mathsf{A},\mathsf{B})) \\
\exists X.\, \mathsf{B} & =_{\mathrm{def}} \forall Y.\, (\forall X.\, (\mathsf{B} \to Y)) \to Y & (Y \notin \mathrm{ftv}(\mathsf{B})) \\
\mu X.\, \mathsf{B} & =_{\mathrm{def}} \forall X.\, (\mathsf{B} \to X) \to X & (X \text{ +ve in } \mathsf{B}) \\
\nu X.\, \mathsf{B} & =_{\mathrm{def}} \exists X.\, (X \to \mathsf{B}) \times X & (X \text{ +ve in } \mathsf{B}) \\
\exists \underline{X}.\, \mathsf{B} & =_{\mathrm{def}} \forall Y.\, (\forall \underline{X}.\, (\mathsf{B} \to Y)) \to Y & (Y \notin \mathrm{ftv}(\mathsf{B}))
\end{array}
$$

Figure 2: Definable value types

second-order λ-calculus, but the presence of the stoup in PE means that in some cases a slight variation of these encodings must be used. A more detailed discussion of this issue appears in [21, Sec. 4].

3. Semantic setting

In the previous section, we appealed to semantic intuition by explaining value types as sets and computation types as algebras for a monad on sets. Unfortunately, this intuition runs into the technical problem that there are no set-theoretic models of polymorphism [33]. However, it was shown by Pitts [25] that set-theoretic models of polymorphism are possible if *intuitionistic* set theory is used rather than ordinary *classical* set theory. We shall exploit this by working with such an intuitionistic set-theoretic model. The advantage of this strategy is that the set-theoretic framework allows the development to concentrate entirely on the difficulties inherent in defining a suitable notion of relational parametricity, which are formidable in themselves, rather than on incidental details specific to a particular concrete model. Our approach results in no loss of generality. All denotational models of relational parametricity of which we are aware can be exhibited as full subcategories of models of intuitionistic set theory.

The intuitionistic set theory we use in this paper is Friedman's Intuitionistic Zermelo-Fraenkel set theory (IZF), which is the established intuitionistic counterpart of classical Zermelo-Fraenkel set theory (ZF). The theory IZF is axiomatized over intuitionistic first-order logic with equality. The axioms of IZF are the usual axioms of classical ZF, except that Collection is taken as an axiom schema instead of Replacement, and Foundation is formulated as a principle of transfinite induction over the membership relation. One reason for assuming the Collection schema is that it is strictly stronger than Replacement under intuitionistic logic. The reformulation of Foundation is required because the usual versions of the axiom imply the Law of Excluded Middle (LEM), whence classical logic. (The Axiom of Choice also implies LEM, and so is not considered.) The naturalness of IZF is underlined by the existence of a wide range of Kripke, sheaf and realizability models. For a detailed summary of the axioms and properties of IZF, see Ščedrov's survey article [36].

Henceforth in this paper, we use IZF as our mathematical meta-theory. To keep matters readable, we work *informally* within IZF, just as in ordinary mathematical practice one works informally in ZF. This approach is deliberately chosen to avoid cluttering the mathematics of the arguments with the formalities of the metatheory. (Nevertheless, when

it is particularly helpful to do so, we shall occasionally remark on technical aspects of the formalization.) In fact, to the casual reader, it will not seem that much out of the ordinary is going on. Given the similarity between the axioms of ZF and IZF, reasoning within IZF feels very much like reasoning within classical ZF. Essentially, the only practical difference is that one has to adhere to the discipline of intuitionistic logic. The reader should try to be sensitive to this issue, because our adherence to intuitionistic logic is essential to the consistency of this paper. Nonetheless, since IZF is a subtheory of ZF, readers who are not familiar with the distinctions between intuitionistic and classical reasoning, should anyway be able to follow the mathematical development. Such readers will, however, have to place their trust in the authors that the reasoning principles of IZF are never violated. For anyone who wishes to learn more about reasoning in intuitionistic set theory, a good starting place is [1].

As is common in set-theoretic reasoning, we shall sometimes have to work with collections of sets that are too "large" to themselves form a set; that is, with proper classes. When working with IZF (as with classical ZF), classes are accommodated by taking them as being represented by formulas: a formula ϕ with distinguished free variable x represents the class $\{x \mid \phi\}$. In practice, it would be a nuisance to always have to work with concrete formulas ϕ. Instead, we shall typically say: "let X be a class then ...", without specifying a particular formula ϕ that represents X. Such reasoning can be understood schematically as being valid relative to any possible formula instantiating X (and, in practice, there may be several different concrete instantiations that satisfy all assumed properties of X). Alternatively, it is possible to view the development as taking place in an extension of the language of set theory with a new unary predicate for every assumed class. This latter viewpoint is slightly more general, since, in models, it allows classes to be collections other than those specified by formulas in the language of set theory. Such mild added generality is natural if one interprets our reasoning in the categorical models of IZF given by *algebraic set theory* [13, 38], where the category of classes is the primary category of interest, and class predicates can be interpreted as objects in such a category. Whichever viewpoint one takes on whether one thinks of the language as extended with class predicates or not, the underlying set theory remains "morally" unchanged, and we shall accordingly continue to refer to it as IZF.

We now begin the technical development within IZF. As discussed above, value types will be modelled as sets. However, it is known that it is not possible to interpret types in the second-order λ-calculus as arbitrary sets [26]. Thus we require a collection of special sets for interpreting types. Such special sets need to be closed under the set-theoretic operations used in the interpretation. Accordingly, we assume that we have a full subcategory $\mathcal{C}$ of the category **Set** of sets that satisfies:

(C1): For any set-indexed family $\{A_i\}_{i\in I}$ of sets in $\mathcal{C}$, the set-theoretic product $\prod_{i\in I} A_i$ is again in $\mathcal{C}$.

(C2): Given $A, B \in \mathcal{C}$ and functions $f, g\colon A \to B$, the equalizer $\{x \in A \mid f(x) = g(x)\}$ is again in $\mathcal{C}$.

In other words, the category $\mathcal{C}$ is small-complete with limits inherited from **Set**. Since function spaces are powers, for any set A and any $B \in \mathcal{C}$, the function space B^A is in $\mathcal{C}$, i.e., $\mathcal{C}$ is an *exponential ideal* of **Set**. In particular, $\mathcal{C}$ is cartesian closed. In addition, we require:

(C3): There is a set $\mathbf{C}$ of objects of $\mathcal{C}$ such that, for any $A \in \mathcal{C}$, there exists $B \in \mathbf{C}$ with $B \cong A$.

(C4): If $A \in \mathcal{C}$ and $A \cong B$ in **Set** then $B \in \mathcal{C}$.

These two properties pull in opposite directions. Property (C3) requires that $\mathcal{C}$ enjoys a smallness constraint, which will be used to interpret polymorphism. Explicitly, (C3) says that $\mathcal{C}$ is weakly equivalent to its small full subcategory on the set of objects $\mathbf{C}$. It is not, however, a small category itself, since (C4) forces $\mathcal{C}$ to have a proper class of objects.

In classical set theory, conditions (C1) and (C3) together imply that every object in $\mathcal{C}$ is either the empty set or a singleton set (cf. Freyd's argument that a weakly small category with small products is a preorder, see [17, Proposition V.2.3]). The reason we work in IZF is that this renders it consistent for there to be a nontrivial category satisfying all of (C1)–(C4). Indeed, it is consistent for the natural numbers to be an object of $\mathcal{C}$. This consistency property derives from the work of Hyland *et. al.* on small-complete small categories [10, 12]. However, our perspective is slightly different. Rather than assuming a small category that is complete only in a restricted technical sense [12, 34], our category $\mathcal{C}$ is assumed to be genuinely complete, but only weakly equivalent to a small category. This approach, which is taken from [35], offers several conveniences. For example, it allows us to assume (C4), which, as well as being a natural repleteness condition on $\mathcal{C}$, makes it easy to show that sets we have defined explicitly are actually in $\mathcal{C}$.

According to our informal explanation of computation types in Section 2, they can be interpreted as Eilenberg-Moore algebras for a monad T on $\mathcal{C}$. For any such monad T, the category $\mathcal{A}$ of algebras comes with a forgetful functor $U\colon \mathcal{A} \to \mathcal{C}$ and the following properties are satisfied.

(A1): U "weakly creates limits" in the following sense. For every diagram Δ in $\mathcal{A}$ and limiting cone $\lim(U(\Delta))$ of $U(\Delta)$ in $\mathcal{C}$, there exists a specified[2] limiting cone $\lim \Delta$ of Δ in $\mathcal{A}$ such that $U(\lim \Delta) = \lim(U(\Delta))$.

(A2): U reflects isomorphisms (i.e., if Uf is an isomorphism in $\mathcal{C}$ then f is an isomorphism in $\mathcal{A}$).

(A3): For objects $\underline{A}, \underline{B}$ of $\mathcal{A}$, the hom-set $\mathcal{A}(\underline{A}, \underline{B})$ is an object of $\mathcal{C}$.

(A4): There exists a set $\mathbf{A}$ of objects of $\mathcal{A}$ such that for every $\underline{A} \in \mathcal{A}$, there exists $\underline{B} \in \mathbf{A}$ with $\underline{B}$ isomorphic to $\underline{A}$.

Lemma 3.1. *Suppose $\mathcal{C}$ satisfies (C1)–(C4) and let T be a monad on $\mathcal{C}$. Then the category $\mathcal{A}$ of Eilenberg-Moore algebras for T and the forgetful functor $U\colon \mathcal{A} \to \mathcal{C}$ satisfy (A1)–(A4).*

Proof. Properties (A1) and (A2) are standard, indeed the forgetful functor creates limits, which implies (A1). Property (A3) holds because $\mathcal{A}(\underline{A}, \underline{B})$ arises as an equalizer in $\mathcal{C}$ of two evident functions $(U\underline{B})^{U\underline{A}} \to (U\underline{B})^{TU\underline{A}}$. For property (A4) define

$$\mathbf{A} = \{(\underline{A}, \xi) \mid \underline{A} \in \mathbf{C}, \text{and } \xi \text{ is an Eilenberg-Moore algebra structure on } \underline{A}\}.$$

□

The reason for identifying (A1)–(A4) is that, in order to interpret the calculus of Section 2, it is sufficient to work with any category $\mathcal{A}$ and functor $U\colon \mathcal{A} \to \mathcal{C}$ satisfying (A1)–(A4) above.[3] Henceforth, we assume this situation.

[2]By a *specified* limiting cone we mean that we are given a (class) function that maps any diagram Δ and limiting cone for $U(\Delta)$ to the required limiting cone in $\mathcal{A}$.

[3]In particular, the weakening of limit creation in (A1) is crucial to the application in [20].

It is convenient to maintain algebraic terminology for the category $\mathcal{A}$. Thus we call the objects of $\mathcal{A}$ *algebras*. By (A1) and (A2), the functor U is faithful, thus we can identify the morphisms $\mathcal{A}(\underline{A}, \underline{B})$ with special functions from $U\underline{A}$ to $U\underline{B}$, which we call *homomorphisms*. We write $\underline{A} \multimap \underline{B}$ for the set of homomorphisms from $\underline{A}$ to $\underline{B}$. (N.B. by (A3) the set $\underline{A} \multimap \underline{B}$ is an object of $\mathcal{C}$.) The notation $\underline{A} \cong^{\circ} \underline{B}$ means $\underline{A}, \underline{B}$ are isomorphic in $\mathcal{A}$.

In Section 4 we interpret the type theory of Section 2 using $U\colon \mathcal{A} \to \mathcal{C}$. In doing so, we formulate relational parametricity using binary relations in the categories $\mathcal{C}$ and $\mathcal{A}$. As usual, these are defined as subobjects of products. First, let us review some basic properties of subobjects in $\mathcal{C}$ and $\mathcal{A}$.

For every object A of $\mathcal{C}$, we write $\mathrm{Sub}_{\mathcal{C}}(A)$ for the set of subobjects of A in the category $\mathcal{C}$. Since the inclusion $\mathcal{C} \hookrightarrow \mathbf{Set}$ preserves limits and hence monomorphisms, this is explicitly defined by:

$$\mathrm{Sub}_{\mathcal{C}}(A) \;=\; \{B \in \mathcal{C} \mid B \subseteq A\}\,.$$

We call the elements of $\mathrm{Sub}_{\mathcal{C}}(A)$ the $\mathcal{C}$-*subsets* of A.

Similarly, we write $\mathrm{Sub}_{\mathcal{A}}(\underline{A})$ for the collection of subobjects of an algebra $\underline{A}$ in $\mathcal{A}$. Because U preserves limits, every mono $\underline{B} \rightarrowtail \underline{A}$ in $\mathcal{A}$ is mapped by U to a mono $U\underline{B} \rightarrowtail U\underline{A}$ in $\mathcal{C}$. Thus, for every $\underline{A} \in \mathcal{A}$, the functor U determines a function $\mathrm{Sub}_{\mathcal{A}}(\underline{A}) \to \mathrm{Sub}_{\mathcal{C}}(U\underline{A})$. The lemma below shows that we can view subobjects of $\underline{A}$ in $\mathcal{A}$ as special subobjects of $U\underline{A}$ in $\mathcal{C}$.

Lemma 3.2. *The function* $\mathrm{Sub}_{\mathcal{A}}(\underline{A}) \to \mathrm{Sub}_{\mathcal{C}}(U\underline{A})$ *preserves and reflects the ordering.*

Proof. We show that it reflects the ordering. Suppose $\underline{B} \rightarrowtail \underline{A}$ and $\underline{C} \rightarrowtail \underline{A}$ represent subobjects of $\underline{A}$ such that the subobject represented by $U\underline{B} \rightarrowtail U\underline{A}$ is smaller than that represented by $U\underline{C} \rightarrowtail U\underline{A}$. Then there exists an f such that the square below is a pullback.

$$\begin{array}{ccc} U\underline{B} & \xrightarrow{\;f\;} & U\underline{C} \\ \| & \lrcorner & \downarrow \\ U\underline{B} & \longrightarrow & U\underline{A} \end{array} \tag{3.1}$$

By (A1) there exists a pullback diagram

$$\begin{array}{ccc} \underline{B'} & \multimap & \underline{C} \\ \downarrow & \lrcorner & \downarrow \\ \underline{B} & \multimap & \underline{A}, \end{array}$$

in $\mathcal{A}$ mapped by U to (3.1), and by (A2) the map $\underline{B'} \multimap \underline{B}$ is an isomorphism, so $\underline{B} \rightarrowtail \underline{A}$ represents a smaller subobject than $\underline{C} \rightarrowtail \underline{A}$. □

We say that $A \subseteq U\underline{A}$ *carries a subalgebra* if it represents a subobject in the image of the map $\mathrm{Sub}_{\mathcal{A}}(\underline{A}) \to \mathrm{Sub}_{\mathcal{C}}(U\underline{A})$ induced by U. In fact, $\mathrm{Sub}_{\mathcal{A}}(\underline{A})$ is given explicitly by:

$$\mathrm{Sub}_{\mathcal{A}}(\underline{A}) = \{B \in \mathcal{C} \,|\, B \subseteq U\underline{A} \text{ and carries a subalgebra of } \underline{A}\}\ .$$

Axiom (A1) gives a way of picking representatives in $\mathcal{A}$ for subalgebras presented by subsets:

Lemma 3.3. *For each $A \in \mathrm{Sub}_{\mathcal{A}}(\underline{A})$ there is a specified algebra $\underline{B}$ and mono $f\colon \underline{B} \rightarrowtail \underline{A}$ in $\mathcal{A}$ such that Uf is the inclusion of A into $U\underline{A}$.*

Proof. Suppose $A \subseteq U\underline{A}$ carries a subalgebra of $\underline{A}$. Then the set

$$\{(\underline{B}, i) \mid \underline{B} \in \mathbf{A}, i\colon \underline{B} \multimap \underline{A} \text{ mono}, U(i) \cong (A \subseteq U\underline{A})\} \tag{3.2}$$

where the last isomorphism is an isomorphism of subobjects, is non-empty. The set (3.2) indexes a diagram in $\mathcal{A}$, and A is a limit in $\mathcal{C}$ of U applied to this diagram. Now, (A1) gives the specified mono projecting to $A \subseteq U\underline{A}$. □

We introduce notation for binary relations. For $A \in \mathcal{C}$, we write Δ_A for the diagonal (identity) relation in $\mathrm{Sub}_{\mathcal{C}}(A \times A)$. Similarly, for $\underline{A} \in \mathcal{A}$, we write $\Delta_{\underline{A}}$ for the diagonal relation on $U\underline{A}$, which is indeed in $\mathrm{Sub}_{\mathcal{A}}(\underline{A}\times\underline{A})$. For $R \in \mathrm{Sub}_{\mathcal{C}}(A\times B)$, we write R^{op} for its opposite relation in $\mathrm{Sub}_{\mathcal{C}}(B \times A)$. Similarly, for $Q \in \mathrm{Sub}_{\mathcal{A}}(\underline{A}\times\underline{B})$, we have $Q^{\mathrm{op}} \in \mathrm{Sub}_{\mathcal{A}}(\underline{B} \times \underline{A})$. For $f\colon A' \to A$ and $g\colon B' \to B$ in $\mathcal{C}$, we write $(f, g)^{-1}R$ for $\{(x, y) \mid (f(x), g(y)) \in R\}$. Notice that if $f\colon \underline{A}' \multimap \underline{A}, g\colon \underline{B}' \multimap \underline{B}$ in $\mathcal{A}$ and $Q \in \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ then $(f, g)^{-1}Q \in \mathrm{Sub}_{\mathcal{A}}(\underline{A}' \times \underline{B}')$.

To formulate relational parametricity, we require two specified collections of *admissible* relations, one $\mathcal{R}_{\mathcal{C}}(A, B) \subseteq \mathrm{Sub}_{\mathcal{C}}(A \times B)$ on objects of $\mathcal{C}$ and one $\mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B}) \subseteq \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ on objects of $\mathcal{A}$. These are required to satisfy:

(R1): For each object A of $\mathcal{C}$ the diagonal relation Δ_A is in $\mathcal{R}_{\mathcal{C}}(A, A)$ and likewise for each object $\underline{A}$ of $\mathcal{A}$ the diagonal $\Delta_{\underline{A}}$ is in $\mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{A})$.

(R2): Admissible relations are closed under reindexing, i.e., if $R \in \mathcal{R}_{\mathcal{C}}(A, B)$ and $f\colon A' \to A$, $g\colon B' \to B$, then $(f, g)^{-1}R \in \mathcal{R}_{\mathcal{C}}(A', B')$ and if $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B})$ and $f\colon \underline{A}' \multimap \underline{A}$, $g\colon \underline{B}' \multimap \underline{B}$, then $(f, g)^{-1}Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}', \underline{B}')$

(R3): For any set of admissible $\mathcal{C}$- (respectively $\mathcal{A}$-)relations on the same pair of objects, the intersection is an admissible $\mathcal{C}$- (respectively $\mathcal{A}$-)relation.

(R4): $\mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B}) \subseteq \mathcal{R}_{\mathcal{C}}(U\underline{A}, U\underline{B})$.

(R1) and (R2) imply that graphs of functions are admissible, i.e., if $f\colon A \to B$ then $\langle f\rangle =_{\mathrm{def}} \{(x, y) \mid f(x) = y\} \in \mathcal{R}_{\mathcal{C}}(A, B)$ and if $g\colon \underline{A} \multimap \underline{B}$ then $\langle g\rangle \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B})$, for $\langle f\rangle = (f, \mathrm{id}_B)^{-1}\Delta_B$ and $\langle g\rangle = (g, \mathrm{id}_{\underline{B}})^{-1}\Delta_{\underline{B}}$. Note also that if $\underline{A}, \underline{B} \in \mathcal{A}$ and $R \subseteq U\underline{A}\times U\underline{B}$ is any subset, then there exists a smallest admissible relation $R^\circ \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B})$ containing R, as we may take R° to be the intersection of all admissible relations containing R.

In many concrete models $\mathcal{R}_{\mathcal{C}}(A, B) = \mathrm{Sub}_{\mathcal{C}}(A \times B)$ and $\mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B}) = \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ will be a natural choice of admissible relations.

Lemma 3.4. *If $\mathcal{C}$ satisfies (C1)–(C4) and $U\colon \mathcal{A} \to \mathcal{C}$ satisfies (A1)–(A4) then the collections $\mathcal{R}_{\mathcal{C}}(A, B) = \mathrm{Sub}_{\mathcal{C}}(A \times B)$ and $\mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B}) = \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ satisfy (R1)–(R4).*

Proof. We just show that $\mathrm{Sub}_{\mathcal{A}}(\underline{A}, \underline{B})$ is closed under intersections. So suppose we are given a set $(Q_i)_{i\in I}$ of subsets in $\mathrm{Sub}_{\mathcal{A}}(\underline{A}, \underline{B})$. We need to show that the subset $\bigcap_i Q_i \subseteq U\underline{A}\times U\underline{B}$ carries a subalgebra of $\underline{A}\times\underline{B}$. Denote for each $i \in I$ by $q_i\colon Q_i' \multimap \underline{A}\times\underline{B}$ the mono in $\mathcal{A}$ above the inclusion $Q_i \subseteq U\underline{A} \times U\underline{B}$ as specified by Lemma 3.3. Then the limit of the diagram given by the q_i as weakly created by U is a subalgebra of $\underline{A}\times\underline{B}$ above $\bigcap_i Q_i \subseteq U\underline{A}\times U\underline{B}$. □

By a *parametric model of PE* we shall mean any category $\mathcal{C}$ satisfying (C1)–(C4), together with a category $\mathcal{A}$ and functor $U\colon \mathcal{A} \to \mathcal{C}$ satisfying (A1)–(A4) and collections $\mathcal{R}_{\mathcal{C}}$ and $\mathcal{R}_{\mathcal{A}}$ satisfying (R1)–(R4) above. The proposition below shows that every monad on $\mathcal{C}$ gives rise to a parametric model of PE. Thus the theory of relational parametricity for PE that we shall develop over such models is applicable to arbitrary computational monads.

Proposition 3.5. *Given $\mathcal{C}$ satisfying (C1)–(C4) and a monad T on $\mathcal{C}$, let $\mathcal{A}$ be the category of algebras for the monad, U the forgetful functor and define $\mathcal{R}_{\mathcal{C}}(A,B) = \mathrm{Sub}_{\mathcal{C}}(A \times B)$ and $\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}) = \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$. This data defines a parametric model of PE.*

Proof. We have already argued above that (A1)–(A4) are satisfied, and (R1)–(R4) are satisfied by Lemma 3.4. □

Notice that the assumption, familiar from the literature on computational monads [22, 23], that the monad T is strong does not need to be included in the above result. This is for the simple reason that our set-theoretic setting renders all monads on $\mathcal{C}$ strong. For any monad T, one defines the strength $t_{A,B} \colon A \times T(B) \to T(A \times B)$ as

$$t_{A,B}(x,y) = T(\langle x, - \rangle)(y)$$

where $\langle x, - \rangle \colon B \to A \times B$ maps y to (x,y). Moreover, this strength is unique because $\mathcal{C}$ has enough points [23, Proposition 3.4].

Although Proposition 3.5 is a useful general result, we comment that some applications of PE require a different choice of model. For example, the application of PE to control in [20] makes crucial use of the permitted flexibility in the definition of model. Here, we briefly describe the steps taken in *op. cit.*, in order to illustrate some of the variations of model construction available. The construction begins with a category $\mathcal{C}$ satisfying (C1)–(C4), together with a chosen object R of $\mathcal{C}$. For technical reasons (see below), the object R is used to isolate the full subcategory $\mathcal{C}_R$ of *R-replete* objects in $\mathcal{C}$, in the sense of [11]. Next, $\mathcal{A}$ together with U are obtained by building $\mathcal{A}$ as a certain carefully defined category equivalent to ${\mathcal{C}_R}^{\mathrm{op}}$, and U as a functor naturally isomorphic to $R^{(-)}$. This situation satisfies (A1)–(A4). The interesting cases are: (A1), which holds by the way $\mathcal{A}$ and U are constructed; and (A2), which holds because we restricted $\mathcal{A}$ to the R-replete objects. Finally, whereas $\mathcal{R}_{\mathcal{C}}(A,B)$ is defined to be $\mathrm{Sub}_{\mathcal{C}}(A \times B)$, it is necessary, for the application to parametricity for control, to define $\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B})$ to be the subset of $\mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ consisting of the $\top\top$-closed relations, in the sense of Pitts [27] (see also [14]), as induced by the diagonal relation Δ_R on R. For full details of this construction, the reader is referred to [20].

One reason that the model construction outlined above departs from the form of model provided by Proposition 3.5 is that, although there is an underlying continuations monad $R^{R^{(-)}}$ present, the category $\mathcal{A}$ is not in general equivalent to the category of algebras for this monad. The usefulness of such more general situations is already familiar from Levy's work on CBPV [15], where the natural adjunction model of control does not involve the Eilenberg-Moore category. One of the strengths of our axiomatic framework is that it is able to accommodate such models.

One of the drawbacks of our framework is that certain convolutions are sometimes necessary in order to construct a model satisfying the properties we require. For example, in the model of control outlined above (and described fully in [20]), awkward steps are taken in order to satisfy properties (A1) and (A2). An arguably preferable approach would be to work with the more natural model in which $\mathcal{A}$ is simply $\mathcal{C}^{\mathrm{op}}$ and U is $R^{(-)}$, as in [15], even though (A1) and (A2) are then violated. This raises the question of whether the awkward properties (A1) and (A2) can be weakened. We shall return to this question in Section 8.

$$
\begin{aligned}
\mathcal{C}[\![X]\!]_\gamma &= \gamma(X)\\
\mathcal{C}[\![\mathsf{B} \to \mathsf{C}]\!]_\gamma &= \mathcal{C}[\![\mathsf{C}]\!]_\gamma{}^{\mathcal{C}[\![\mathsf{B}]\!]_\gamma}\\
\mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma &= \{\pi \in \prod_{A\in\mathbf{C}} \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[A/X]} \mid \forall A, B \in \mathbf{C},\, \forall R \in \mathcal{R}_{\mathcal{C}}(A,B).\, \mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[R/X]}(\pi_A, \pi_B)\}\\
\mathcal{C}[\![\underline{X}]\!]_\gamma &= U(\gamma(\underline{X}))\\
\mathcal{C}[\![\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}]\!]_\gamma &= \mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma \multimap \mathcal{A}[\![\underline{\mathsf{B}}]\!]_\gamma\\
\mathcal{C}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\gamma &= \{\kappa \in \prod_{\underline{A}\in\mathbf{A}} \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\underline{A}/\underline{X}]} \mid \forall \underline{A}, \underline{B} \in \mathbf{A},\, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B}).\, \mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[Q/\underline{X}]}(\kappa_{\underline{A}}, \kappa_{\underline{B}})\}\ .
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{A}[\![\mathsf{B} \to \underline{\mathsf{A}}]\!]_\gamma &= \mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma{}^{\mathcal{C}[\![\mathsf{B}]\!]_\gamma}\\
\mathcal{A}[\![\forall X.\, \underline{\mathsf{A}}]\!]_\gamma &= \{\pi \in \prod_{A\in\mathbf{C}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[A/X]} \mid \forall A, B \in \mathbf{C},\, \forall R \in \mathcal{R}_{\mathcal{C}}(A,B).\, \mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_\gamma[R/X]}(\pi_A, \pi_B)\}\\
\mathcal{A}[\![\underline{X}]\!]_\gamma &= \gamma(\underline{X})\\
\mathcal{A}[\![\forall \underline{X}.\, \underline{\mathsf{A}}]\!]_\gamma &= \{\kappa \in \prod_{\underline{A}\in\mathbf{A}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[\underline{A}/\underline{X}]} \mid \forall \underline{A}, \underline{B} \in \mathbf{A},\, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B}).\, \mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_\gamma[Q/\underline{X}]}(\kappa_{\underline{A}}, \kappa_{\underline{B}})\}\ .
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{R}[\![X]\!]_\rho(x_1,x_2) &\Leftrightarrow \rho_{\mathcal{R}}(X)(x_1,x_2)\\
\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho(f_1,f_2) &\Leftrightarrow \forall x_1 \in \mathcal{C}[\![\mathsf{B}]\!]_{\rho_1}, x_2 \in \mathcal{C}[\![\mathsf{B}]\!]_{\rho_2}.\, \mathcal{R}[\![\mathsf{B}]\!]_\rho(x_1,x_2) \implies \mathcal{R}[\![\mathsf{C}]\!]_\rho(f_1(x_1), f_2(x_2))\\
\mathcal{R}[\![\forall X.\, \mathsf{B}]\!]_\rho(\pi_1,\pi_2) &\Leftrightarrow \forall A_1, A_2 \in \mathbf{C}, \forall R \in \mathcal{R}_{\mathcal{C}}(A_1,A_2).\, \mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}((\pi_1)_{A_1}, (\pi_2)_{A_2})\\
\mathcal{R}[\![\underline{X}]\!]_\rho(x_1,x_2) &\Leftrightarrow \rho_{\mathcal{R}}(\underline{X})(x_1,x_2)\\
\mathcal{R}[\![\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}]\!]_\rho(h_1,h_2) &\Leftrightarrow \forall x_1 \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_{\rho_1}, x_2 \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_{\rho_2}.\, \mathcal{R}[\![\underline{\mathsf{A}}]\!]_\rho(x_1,x_2) \implies \mathcal{R}[\![\underline{\mathsf{B}}]\!]_\rho(h_1(x_1), h_2(x_2))\\
\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\rho(\kappa_1,\kappa_2) &\Leftrightarrow \forall \underline{A}_1, \underline{A}_2 \in \mathbf{A}, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}_1, \underline{A}_2).\, \mathcal{R}[\![\mathsf{B}]\!]_{\rho[Q/\underline{X}]}((\kappa_1)_{\underline{A}_1}, (\kappa_2)_{\underline{A}_2})\ .
\end{aligned}
$$

Figure 3: Interpretation of Types

4. Interpreting the calculus

In this section we interpret PE in any parametric model as defined in Section 3. As adumbrated there, a value type B will be interpreted as a set $\mathcal{C}[\![\mathsf{B}]\!]$ in $\mathcal{C}$, and a computation type $\underline{\mathsf{A}}$ will be interpreted as an algebra $\mathcal{A}[\![\underline{\mathsf{A}}]\!]$. Since every computation type $\underline{\mathsf{A}}$ is also a value type, it is given two interpretations, and we shall ensure that these are related by $U(\mathcal{A}[\![\underline{\mathsf{A}}]\!]) = \mathcal{C}[\![\underline{\mathsf{A}}]\!]$. In order to incorporate relational parametricity, we shall also give a second interpretation of a value type B as an admissible $\mathcal{C}$-relation $\mathcal{R}[\![\mathsf{B}]\!]$. In the special case of a computation type $\underline{\mathsf{A}}$, it will hold automatically that $\mathcal{R}[\![\underline{\mathsf{A}}]\!]$ is also an admissible $\mathcal{A}$-relation.

Given a set of type variables Θ, a Θ-*environment* is a function γ mapping every value-type variable $X \in \Theta$ to an object $\gamma(X)$ of $\mathcal{C}$, and every computation-type variable $\underline{X} \in \Theta$ to an object $\gamma(\underline{X})$ of $\mathcal{A}$. A *relational* Θ-*environment* is a tuple $\rho = (\rho_1, \rho_2, \rho_{\mathcal{R}})$, where: ρ_1, ρ_2 are Θ-environments; for every value-type variable $X \in \Theta$,

$$\rho_{\mathcal{R}}(X) \in \mathcal{R}_{\mathcal{C}}(\rho_1(X), \rho_2(X))\ ;$$

and, for every computation-type variable $\underline{X} \in \Theta$,

$$\rho_{\mathcal{R}}(\underline{X}) \in \mathcal{R}_{\mathcal{A}}(\rho_1(\underline{X}), \rho_2(\underline{X})) \ .$$

For each value type $\mathsf{B}(\Theta)$ (i.e., type B with $\mathrm{ftv}(\mathsf{B}) \subseteq \Theta$) and Θ-environment γ, we define an object $\mathcal{C}[\![\mathsf{B}]\!]_\gamma$ of $\mathcal{C}$; and, for each computation type $\underline{\mathsf{A}}(\Theta)$ and Θ-environment γ, we define an object $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma$ of $\mathcal{A}$. Interdependently with the above, for each value type $\mathsf{B}(\Theta)$ and relational Θ-environment ρ, we define an admissible $\mathcal{C}$-relation $\mathcal{R}[\![\mathsf{B}]\!]_\rho \in \mathcal{R}_{\mathcal{C}}(\mathcal{C}[\![\mathsf{B}]\!]_{\rho_1}, \mathcal{C}[\![\mathsf{B}]\!]_{\rho_2})$. The definitions are given in Figure 3. In these definitions, the products and powers used in the definition of $\mathcal{C}[\![\mathsf{B}]\!]_\gamma$ are the ones in $\mathcal{C}$, and those used in the definition of $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma$ are those in $\mathcal{A}$, as (weakly) created by U. We write Δ_γ for the relational Θ-environment that maps X (resp. $\underline{X}$) to $\Delta_{\gamma(X)}$ (resp. $\Delta_{\gamma(\underline{X})}$). We also use an obvious notation for update of environments. The algebras defined by $\mathcal{A}[\![\forall Y.\underline{\mathsf{A}}]\!]_\gamma$ and $\mathcal{A}[\![\forall \underline{X}.\underline{\mathsf{A}}]\!]_\gamma$ are the canonical algebras carried by the subsets of the product algebras.

Proposition 4.1. *$\mathcal{C}[\![\mathsf{B}]\!]_\gamma$, $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma$ and $\mathcal{R}[\![\mathsf{B}]\!]_\rho$ are well defined by Figure 3. Further, for every computation type $\underline{\mathsf{A}}$, it holds that $\mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma = U(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma)$ and $\mathcal{R}[\![\underline{\mathsf{A}}]\!]_\rho \in \mathcal{R}_{\mathcal{A}}(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\rho_1}, \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\rho_2})$.*

Proof. The proof of well definedness is by induction over the structure of types. We focus first on showing that the relational interpretation of types defines admissible relations. Notice first that the relation $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ can be rewritten as

$$\bigcap_{(x_1,x_2)\in\mathcal{R}[\![\mathsf{B}]\!]_\rho} (\mathrm{ev}_{x_1}, \mathrm{ev}_{x_2})^{-1}\mathcal{R}[\![\mathsf{C}]\!]_\rho$$

where ev_{x_1} denotes the map from $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_{\rho_1}$ to $\mathcal{R}[\![\mathsf{C}]\!]_{\rho_1}$ given by evaluation at x_1, and ev_{x_2} is defined likewise. For value types B, C it follows that $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ is an admissible $\mathcal{C}$ relation from the induction hypothesis and (R2) and (R3). If C is a computation type, $\mathsf{B} \to \mathsf{C}$ becomes a computation type and we must check that $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ is an admissible $\mathcal{A}$ relation. Since the object $\mathcal{A}[\![\mathsf{B} \to \mathsf{C}]\!]_{\rho_1}$ is defined as a product $\mathcal{A}[\![\mathsf{C}]\!]_{\rho_1}^{\mathcal{C}[\![\mathsf{B}]\!]_{\rho_1}}$ in $\mathcal{A}$ and the evaluation map ev_{x_1} is the projection, it is a homomorphism. So again $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ being admissible follows from the induction hypothesis and (R2), (R3). The proof of the other induction cases are similar.

To prove well definedness of $\mathcal{A}[\![\forall X.\,\underline{\mathsf{A}}]\!]_\gamma$ notice first that the formula in Figure 3 defines an element in $\mathrm{Sub}_{\mathcal{A}}(\prod_{A\in\mathbf{C}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[A/X]})$ since it can be exhibited as the intersection

$$\bigcap_{A,B\in\mathbf{C},R\in\mathcal{R}_{\mathcal{C}}(A,B)} (p_A, p_B)^{-1}\mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_\gamma[R/X]} \tag{4.1}$$

where p_A, p_B are the projections from the product $\prod_{A\in\mathbf{C}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[A/X]}$. The projections are homomorphisms since the product is taken in the category $\mathcal{A}$ and thus, since $\mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_\gamma[R/X]}$ is an $\mathcal{A}$-subobject by induction hypothesis, (4.1) defines an $\mathcal{A}$-subobject. We define $\mathcal{A}[\![\forall X.\underline{\mathsf{A}}]\!]_\gamma$ to be the specified $\mathcal{A}$ object representing the subset as given by Lemma 3.3, thus defining $\mathcal{A}[\![\forall X.\,\underline{\mathsf{A}}]\!]_\gamma$ up to identity and not just up to isomorphism. □

We include some basic lemmata about the type interpretation without proof.

Lemma 4.2. *Suppose γ is a Θ-environment and ρ is a relational Θ environment.*
(1) *If $\mathsf{B}(\Theta, X)$ and $\mathsf{A}(\Theta)$ then*

$$\mathcal{C}[\![\mathsf{B}[\mathsf{A}/X]]\!]_\gamma = \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\mathcal{C}[\![\mathsf{A}]\!]_\gamma/X]}$$
$$\mathcal{R}[\![\mathsf{B}[\mathsf{A}/X]]\!]_\rho = \mathcal{R}[\![\mathsf{B}]\!]_{\rho[\mathcal{R}[\![\mathsf{A}]\!]_\rho/X]}$$

(2) *If* $\mathsf{B}(\Theta, \underline{X})$ *and* $\underline{\mathsf{A}}(\Theta)$ *then*

$$\mathcal{C}[\![\mathsf{B}[\underline{\mathsf{A}}/\underline{X}]]\!]_\gamma = \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma/\underline{X}]}$$
$$\mathcal{R}[\![\mathsf{B}[\underline{\mathsf{A}}/\underline{X}]]\!]_\rho = \mathcal{R}[\![\mathsf{B}]\!]_{\rho[\mathcal{R}[\![\underline{\mathsf{A}}]\!]_\rho/\underline{X}]}$$

(3) *If* $\underline{\mathsf{B}}(\Theta, X)$ *and* $\mathsf{A}(\Theta)$ *then*

$$\mathcal{A}[\![\underline{\mathsf{B}}[\mathsf{A}/X]]\!]_\gamma = \mathcal{A}[\![\underline{\mathsf{B}}]\!]_{\gamma[\mathcal{C}[\![\mathsf{A}]\!]_\gamma/X]}$$

(4) *If* $\underline{\mathsf{B}}(\Theta, \underline{X})$ *and* $\underline{\mathsf{A}}(\Theta)$ *then*

$$\mathcal{A}[\![\underline{\mathsf{B}}[\underline{\mathsf{A}}/\underline{X}]]\!]_\gamma = \mathcal{A}[\![\underline{\mathsf{B}}]\!]_{\gamma[\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma/\underline{X}]}$$

Lemma 4.3. *For all types* $\mathsf{A}(\Theta)$ *and any* Θ*-environment* γ *the relations* $\mathcal{R}[\![\mathsf{A}]\!]_\gamma{}^{\mathrm{op}}$ *and* $\mathcal{R}[\![\mathsf{A}]\!]_{\gamma^{\mathrm{op}}}$ *are equal, where* γ^{op} *is the environment obtained by composing* γ *with the function* $(-)^{\mathrm{op}}$*.*

Lemma 4.4 (Identity extension)**.** *For any type* $\mathsf{B}(\Theta)$ *and* Θ*-environment* γ*, it holds that* $\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma} = \Delta_{\mathcal{C}[\![\mathsf{B}]\!]_\gamma}$*.*

The above lemmata are all easily proved by induction on types.

The interpretations of polymorphic types have been defined by taking products over the sets $\mathbf{C}, \mathbf{A}$ respectively, but for the interpretation of terms below, it is crucial that we can define projections out these products for every A in $\mathcal{C}$ (respectively $\underline{B}$ in $\mathcal{A}$) and not just for those objects in the sets $\mathbf{C}, \mathbf{A}$. Essentially, we would like to be able to treat these polymorphic types as if they had been defined using products over the classes of objects of $\mathcal{C}$ and $\mathcal{A}$, even though set theory does not allow us to define such large products. It is a pleasing fact that restriction to the parametric elements of the products allows us to do just that, as the sequence of results from Proposition 4.5 to Lemma 4.10 below establishes. The idea essentially goes back to [35], and was used in [18] to construct a model of parametric polymorphism in the sense of fibered category theory.

To formulate the first result, we define a *morphism* from Θ-environments γ to another γ' to be a family $\mathbf{f}$ of functions indexed by type variables in Θ satisfying: for every value-type variable $X \in \Theta$, the function $\mathbf{f}_X$ is a function from $\gamma(X)$ to $\gamma'(X)$; and, for every computation-type variable $\underline{X} \in \Theta$, the function $\mathbf{f}_{\underline{X}}$ is a homomorphism from $\gamma(\underline{X})$ to $\gamma'(\underline{X})$. Morphisms of Θ-environments form a category under pointwise composition, and a Θ-environment *isomorphism* is just an isomorphism in this category. Given a Θ-environment morphism $\mathbf{f}$ from γ to γ', we write $\langle \mathbf{f} \rangle$ for the relational Θ-environment with $\langle \mathbf{f} \rangle_1 = \gamma$, and $\langle \mathbf{f} \rangle_2 = \gamma'$ and $\langle \mathbf{f} \rangle_{\mathcal{R}}(X) = \langle \mathbf{f}_X \rangle$ and $\langle \mathbf{f} \rangle_{\mathcal{R}}(\underline{X}) = \langle \mathbf{f}_{\underline{X}} \rangle$. Also, given a Θ-environment, γ, we write $\mathbf{x} \in \gamma$ for a family of elements indexed by type variables in Θ satisfying: for every value-type variable $X \in \Theta$, it holds that $\mathbf{x}_X \in \gamma(X)$ and, for every computation-type variable $\underline{X} \in \Theta$, it holds that $\mathbf{x}_{\underline{X}} \in U(\gamma(\underline{X}))$. Given a Θ-environment morphism $\mathbf{f} \colon \gamma \to \gamma'$ and $\mathbf{x} \in \gamma$, we write $\mathbf{f}(\mathbf{x})$ for the evident pointwise function application, which is an element of γ'. Moreover, given a relational Θ-environment ρ, and elements $\mathbf{x_1} \in \rho_1$ and $\mathbf{x_2} \in \rho_2$, we write $\rho_{\mathcal{R}}(\mathbf{x_1}, \mathbf{x_2})$ to mean that: for every $X \in \Theta$, it holds that $\rho_{\mathcal{R}}(X)(\mathbf{x_1}_X, \mathbf{x_2}_X)$; and, for every $\underline{X} \in \Theta$, it holds that $\rho_{\mathcal{R}}(\underline{X})(\mathbf{x_1}_{\underline{X}}, \mathbf{x_2}_{\underline{X}})$.

Proposition 4.5 (Groupoid action)**.** *For any type* $\mathsf{C}(\Theta)$*, any two* Θ*-environments* γ*,* γ'*, and any* Θ*-environment isomorphism* $\mathbf{i} \colon \gamma \to \gamma'$*, there exists a unique isomorphism*

$$\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i}) \colon \mathcal{C}[\![\mathsf{C}]\!]_\gamma \to \mathcal{C}[\![\mathsf{C}]\!]_{\gamma'}$$

such that

$$\mathcal{R}[\![\mathsf{C}]\!]_{\langle \mathbf{i} \rangle} \;=\; \langle \mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i}) \rangle \;\; .$$

Moreover, if C *is a computation type then* $\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i})$ *is a homomorphism from* $\mathcal{A}[\![\mathsf{C}]\!]_\gamma$ *to* $\mathcal{A}[\![\mathsf{C}]\!]_{\gamma'}$.

Furthermore, given relational Θ*-environments* ρ, ρ', *and given* Θ*-environment isomorphisms* $\mathbf{i_1} \colon \rho_1 \to \rho_1'$ *and* $\mathbf{i_2} \colon \rho_2 \to \rho_2'$*; if, for all* $\mathbf{x_1} \in \rho_1, \mathbf{x_2} \in \rho_2$,

$$\rho_R(\mathbf{x_1}, \mathbf{x_2}) \implies \rho_R'(\mathbf{i_1}(\mathbf{x_1}), \mathbf{i_2}(\mathbf{x_2})) \;\; ,$$

then, for all $x_1 \in \mathcal{C}[\![\mathsf{C}]\!]_{\rho_1}, x_2 \in \mathcal{C}[\![\mathsf{C}]\!]_{\rho_2}$, *we have:*

$$\mathcal{R}[\![\mathsf{C}]\!]_\rho(x_1, x_2) \implies \mathcal{R}[\![\mathsf{C}]\!]_{\rho'}(\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i_1})(x_1), \mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i_2})(x_2)) \;\; .$$

Proof. By induction on the structure of the type C. We consider two cases.

If C is $\mathsf{A} \to \mathsf{B}$ then the induction hypothesis gives isomorphisms $\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i}) \colon \mathcal{C}[\![\mathsf{A}]\!]_\gamma \to \mathcal{C}[\![\mathsf{A}]\!]_{\gamma'}$ and $\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}) \colon \mathcal{C}[\![\mathsf{B}]\!]_\gamma \to \mathcal{C}[\![\mathsf{B}]\!]_{\gamma'}$. Using that $\mathcal{R}[\![\mathsf{A}]\!]_{\langle \mathbf{i} \rangle} = \langle \mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i}) \rangle$ and $\mathcal{R}[\![\mathsf{B}]\!]_{\langle \mathbf{i} \rangle} = \langle \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}) \rangle$, one calculates that

$$\mathcal{R}[\![\mathsf{A} \to \mathsf{B}]\!]_{\langle \mathbf{i} \rangle} \;=\; \langle f \mapsto \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}) \circ f \circ (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i}))^{-1} \rangle \;\; ,$$

so we have:

$$\mathrm{gpd}[\![\mathsf{A} \to \mathsf{B}]\!](\mathbf{i}) \;=\; f \mapsto \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}) \circ f \circ (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i}))^{-1} \;\; ,$$

which obviously is an isomorphism. Further, C is a computation type just when B is, in which case we must show that $\mathrm{gpd}[\![\mathsf{A} \to \mathsf{B}]\!](\mathbf{i})$, as defined above, is a homomorphism. By definition $\mathcal{A}[\![\mathsf{A} \to \mathsf{B}]\!]_{\gamma'}$ is a $\mathcal{C}[\![\mathsf{A}]\!]_{\gamma'}$-fold product of $\mathcal{A}[\![\mathsf{B}]\!]_{\gamma'}$ by itself as taken in $\mathcal{A}$, and each evaluation map ev_x, for $x \in \mathcal{C}[\![\mathsf{A}]\!]_{\gamma'}$, is a projection. It suffices to show that for each $x \in \mathcal{C}[\![\mathsf{A}]\!]_{\gamma'}$ the composite $\mathrm{ev}_x \circ \mathrm{gpd}[\![\mathsf{A} \to \mathsf{B}]\!](\mathbf{i})$ is a homomorphism. But

$$\begin{aligned} \mathrm{ev}_x \circ \mathrm{gpd}[\![\mathsf{A} \to \mathsf{B}]\!](\mathbf{i})(f) &= \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}) \circ f \circ (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i}))^{-1}(x) \\ &= \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}) \circ \mathrm{ev}_{(\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i}))^{-1}(x)}(f) \end{aligned}$$

and $\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i})$ is a homomorphism by induction hypothesis, and evaluation maps are homomorphisms because they are projections out of a product taken in $\mathcal{A}$.

For the second half of the proposition, given isomorphisms $\mathbf{i_1} \colon \rho_1 \to \rho_1'$ and $\mathbf{i_2} \colon \rho_2 \to \rho_2'$ as in the hypothesis, we must show that if $\mathcal{R}[\![\mathsf{A} \to \mathsf{B}]\!]_\rho(f_1, f_2)$ and $\mathcal{R}[\![\mathsf{A}]\!]_{\rho'}(x_1, x_2)$ then

$$\mathcal{R}[\![\mathsf{B}]\!]_{\rho'}(\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i_1}) \circ f_1 \circ (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}))^{-1}(x_1), \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i_2}) \circ f_2 \circ (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_2}))^{-1}(x_2)) \quad (4.2)$$

Note first that $(\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}))^{-1} = \mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}^{-1})$ because

$$\begin{aligned} \langle (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}))^{-1} \rangle &= \langle \mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}) \rangle^{\mathrm{op}} \\ &= \mathcal{R}[\![\mathsf{A}]\!]_{\langle \mathbf{i_1} \rangle}{}^{\mathrm{op}} \\ &= \mathcal{R}[\![\mathsf{A}]\!]_{\langle \mathbf{i_1} \rangle^{\mathrm{op}}} \\ &= \mathcal{R}[\![\mathsf{A}]\!]_{\langle \mathbf{i_1}^{-1} \rangle} \\ &= \langle \mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}^{-1}) \rangle \end{aligned}$$

where we have used Lemma 4.3. Similarly $(\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_2}))^{-1} = \mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_2}^{-1})$. So by the induction hypothesis, under the assumptions stated above

$$\mathcal{R}[\![\mathsf{A}]\!]_\rho((\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}))^{-1}(x_1), (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_2}))^{-1}(x_2))$$

and so also

$$\mathcal{R}[\![\mathsf{B}]\!]_\rho(f_1 \circ (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_1}))^{-1}(x_1), f_2 \circ (\mathrm{gpd}[\![\mathsf{A}]\!](\mathbf{i_2}))^{-1}(x_2))$$

from which we conclude (4.2) by a second application of the induction hypothesis.

We define $\mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})$ by the formula

$$(\mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})(\kappa))_{\underline{A}} = \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}])(\kappa_{\underline{A}}) \ .$$

to see that this is well defined we must show that if $\underline{A}, \underline{C} \in \mathbf{A}$ and $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{C})$ then

$$\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_{\gamma'}[Q/\underline{X}]}(\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}])(\kappa_{\underline{A}}), \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{C}}/\underline{X}])(\kappa_{\underline{C}})) \ . \tag{4.3}$$

But since $\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[Q/\underline{X}]}(\kappa_{\underline{A}}, \kappa_{\underline{C}})$ and since the pair $(\mathbf{i}[\mathrm{id}_{\underline{A}}], \mathbf{i}[\mathrm{id}_{\underline{C}}])$ maps pairs related in $\Delta_\gamma[Q/\underline{X}]$ to pairs related in $\Delta_{\gamma'}[Q/\underline{X}]$ the induction hypothesis implies (4.3).

To show $\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_{\langle \mathbf{i} \rangle} = \langle \mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i}) \rangle$, first suppose that $\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_{\langle \mathbf{i} \rangle}(\kappa_1, \kappa_2)$. Then $\mathcal{R}[\![\mathsf{B}]\!]_{\langle \mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}] \rangle}((\kappa_1)_{\underline{A}}, (\kappa_2)_{\underline{A}})$ for all $\underline{A}$ and so by induction hypothesis $((\kappa_1)_{\underline{A}}, (\kappa_2)_{\underline{A}})$ is in $\langle \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}]) \rangle$ which implies $\mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})(\kappa_1) = \kappa_2$. Suppose on the other hand that $\mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})(\kappa_1) = \kappa_2$. Then

$$\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_{\gamma'}[Q/\underline{X}]}((\mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})(\kappa_1))_{\underline{A}}, (\kappa_2)_{\underline{C}})$$

for all $\underline{A}, \underline{C} \in \mathbf{A}$ and $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{C})$, i.e.,

$$\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_{\gamma'}[Q/\underline{X}]}(\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}])((\kappa_1)_{\underline{A}}), (\kappa_2)_{\underline{C}}) \ . \tag{4.4}$$

The pair $(\mathbf{i}^{-1}[\mathrm{id}_{\underline{C}}/\underline{X}], \mathrm{id}_{\gamma'}[\mathrm{id}_{\underline{A}}/\underline{X}])$ maps pairs related in $\Delta_{\gamma'}[Q/\underline{X}]$ to pairs related in $\langle \mathbf{i} \rangle[Q/\underline{X}]$, and so by induction hypothesis, the pair $(\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}^{-1}[\mathrm{id}_{\underline{A}}/\underline{X}]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\gamma'}[\mathrm{id}_{\underline{A}}/\underline{X}]))$ maps pairs related in $\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_{\gamma'}[Q/\underline{X}]}$ to pairs related in $\mathcal{R}[\![\mathsf{B}]\!]_{\langle \mathbf{i} \rangle[Q/\underline{X}]}$. As above, one can show that

$$\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}^{-1}[\mathrm{id}_{\underline{A}}/\underline{X}]) = (\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}]))^{-1}$$

and using Lemma 4.3 also $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\gamma'}[\mathrm{id}_{\underline{A}}/\underline{X}]) = \mathrm{id}_{\mathcal{C}[\![\mathsf{B}]\!]_{\gamma'}[\underline{A}/\underline{X}]}$ and so by (4.4) we conclude

$$\mathcal{R}[\![\mathsf{B}]\!]_{\langle \mathbf{i} \rangle[Q/\underline{X}]}((\kappa_1)_{\underline{A}}, (\kappa_2)_{\underline{C}}) \ .$$

Since this holds for all $\underline{A}, \underline{C} \in \mathbf{A}$ and $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{C})$ this implies $\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_{\langle \mathbf{i} \rangle}(\kappa_1, \kappa_2)$. In conclusion we have shown $\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_{\langle \mathbf{i} \rangle} = \langle \mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i}) \rangle$.

The type $\forall \underline{X}.\, \mathsf{B}$ is a computation type exactly when B is, and in this case we must show that $\mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})$ is a homomorphism. Similarly to the case of function spaces, since $\mathcal{A}[\![\forall \underline{X}.\, \mathsf{B}]\!]_{\gamma'}$ is constructed as a limit in $\mathcal{A}$ it suffices to show that each composite $p_{\underline{A}} \circ \mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})$ is a homomorphism, where $p_{\underline{A}}$ is the projection defined as $p_{\underline{A}}(\kappa) = \kappa_{\underline{A}}$. Since

$$\begin{aligned} p_{\underline{A}} \circ \mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i})(\kappa) &= \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}])(\kappa_{\underline{A}}) \\ &= \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i}[\mathrm{id}_{\underline{A}}/\underline{X}]) \circ p_{\underline{A}}(\kappa) \end{aligned}$$

this follows by the induction hypothesis.

For the last part of the proposition, suppose the pair $(\mathbf{i_1}, \mathbf{i_2})$ maps pairs related in ρ to pairs related in ρ', and suppose $\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_{\rho}(\kappa_1, \kappa_2)$. We must show that

$$\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_{\rho'}(\mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i_1})(\kappa_1), \mathrm{gpd}[\![\forall \underline{X}.\, \mathsf{B}]\!](\mathbf{i_2})(\kappa_2)) \ ,$$

i.e., we must show that for any $\underline{A}, \underline{C} \in \mathbf{A}$, $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{C})$

$$\mathcal{R}[\![\mathsf{B}]\!]_{\rho'[Q/\underline{X}]}(\mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i_1}[\mathrm{id}_{\underline{A}}/\underline{X}])((\kappa_1)_{\underline{A}}), \mathrm{gpd}[\![\mathsf{B}]\!](\mathbf{i_2}[\mathrm{id}_{\underline{C}}/\underline{X}])((\kappa_2)_{\underline{C}})) \ .$$

Since the pair $(\mathbf{i_1}[\mathrm{id}_{\underline{A}}/\underline{X}], \mathbf{i_2}[\mathrm{id}_{\underline{C}}/\underline{X}])$ maps pairs related in $\rho[Q/\underline{X}]$ to pairs related in $\rho'[Q/\underline{X}]$ this follows from the induction hypothesis. □

Corollary 4.6. *The mapping of isomorphisms between Θ-environments, $\mathbf{i}$, to* $\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i})$ *is functorial.*

Proof. Preservation of identities is Lemma 4.4. For preservation of composition, suppose $\mathbf{i}\colon \rho \to \rho'$ and $\mathbf{j}\colon \rho' \to \rho''$. If $\langle \mathbf{i}\rangle(\mathbf{x},\mathbf{y})$ then $\langle \mathbf{j}\circ\mathbf{i}\rangle(\mathbf{x},\mathbf{j}(\mathbf{y}))$ so by Proposition 4.5, if $\mathcal{R}[\![\mathsf{C}]\!]_{\langle \mathbf{i}\rangle}(x,y)$ then $\mathcal{R}[\![\mathsf{C}]\!]_{\langle \mathbf{j}\circ\mathbf{i}\rangle}(x,\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{j})(y))$. Since $\mathcal{R}[\![\mathsf{C}]\!]_{\langle \mathbf{i}\rangle}(x,\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i})(x))$, we conclude

$$\mathcal{R}[\![\mathsf{C}]\!]_{\langle \mathbf{j}\circ\mathbf{i}\rangle}(x,\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{j})\circ\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i})(x))$$

for all x, i.e., $\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{j})\circ\mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{i}) = \mathrm{gpd}[\![\mathsf{C}]\!](\mathbf{j}\circ\mathbf{i})$ as desired. □

Corollary 4.7. *For any type $\mathsf{B}(\Theta, X)$, relational Θ-environment ρ, any relation R in $\mathcal{R}_{\mathcal{C}}(A,C)$, and any pair of isomorphisms $i\colon A' \to A$, $j\colon C' \to C$*

$$\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]} = (\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i/X]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j/X]))^{-1}\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]} \ .$$

Similarly for any type $\mathsf{B}(\Theta, \underline{X})$, relational Θ-environment ρ, any relation $R \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{C})$, and any pair of isomorphisms $i\colon \underline{A}' \multimap \underline{A}$, $j\colon \underline{C}' \to \underline{C}$

$$\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/\underline{X}]} = (\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i/\underline{X}]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j/\underline{X}]))^{-1}\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/\underline{X}]} \ .$$

Proof. We just prove the first part. Since the pair (i,j) maps pairs related in $(i,j)^{-1}R$ to pairs related in R, by Proposition 4.5 the pair $(\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i/X]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j/X]))$ maps pairs related in $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]}$ to pairs related in $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}$. This means that

$$\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]} \subseteq (\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i/X]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j/X]))^{-1}\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]} \ . \tag{4.5}$$

Since $R = (i^{-1}, j^{-1})^{-1}(i,j)^{-1}R$ we can apply the above to the pair (i^{-1}, j^{-1}) and obtain

$$\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]} \subseteq (\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i^{-1}/X]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j^{-1}/X]))^{-1}\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]}$$

from which we conclude

$$(\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i/X]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j/X]))^{-1}\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]} \subseteq \mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]} \ . \tag{4.6}$$

The corollary is now the collected statement of (4.5) and (4.6). □

Now, for any set A in $\mathcal{C}$, let $C \in \mathbf{C}$ be such that $C \cong A$ by way of the isomorphism $i\colon C \to A$. Using the groupoid action defined above, we have $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i/X])(\pi_C) \in \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[A/X]}$. Similarly, for any algebra $\underline{A}$ in $\mathcal{A}$, let $\underline{C} \in \mathbf{A}$ be such that $\underline{C} \cong^{\circ} \underline{A}$ by way of $j\colon \underline{C} \multimap \underline{A}$. Then we have $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[j/\underline{X}])(\kappa_{\underline{C}}) \in \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\underline{A}/\underline{X}]}$.

Lemma 4.8. *For $\pi \in \mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma$ and A in $\mathcal{C}$:*

(1) *The value* $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i/X])(\pi_C)$ *is independent of the choice of C and i.*
(2) *If $A \in \mathbf{C}$ then* $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i/X])(\pi_C) = \pi_A$.

Similarly, for $\kappa \in \mathcal{C}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\gamma$ and $\underline{A} \in \mathcal{A}$:

(3) *The value* $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[j/\underline{X}])(\kappa_{\underline{C}})$ *is independent of the choice of $\underline{C}$ and j.*
(4) *If $\underline{A} \in \mathbf{A}$ then* $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[j/\underline{X}])(\kappa_{\underline{C}}) = \kappa_{\underline{A}}$.

Proof. We prove 1. Suppose $i\colon C \to A, i'\colon C' \to A$ are isomorphisms. We must show that $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i/X])(\pi_C) = \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i'/X])(\pi_{C'})$. By the parametricity condition in the definition of $\mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma$, $\mathcal{C}[\![\mathsf{B}]\!]_{\Delta_\gamma[\langle i'^{-1}\circ i\rangle/X]}(\pi_C, \pi_{C'})$, which means that

$$\langle \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i'^{-1}\circ i/X])\rangle(\pi_C, \pi_{C'}) \, .$$

Now by definition of graph relations and functoriality of the groupoid action this implies

$$\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i/X])(\pi_C) = \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i'/X])(\pi_{C'})$$

as desired. Item 2 is an immediate consequence: use the identity on A for i. □

The above lemma justifies introducing the following very useful notation. Given π in $\mathcal{C}[\![\forall X.\mathsf{B}]\!]_\gamma$, then, for any A in $\mathcal{C}$, we write $\pi(A)$ for $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[i/X])(\pi_C)$, where $i\colon C \to A$ is an isomorphism and $C \in \mathbf{C}$. Similarly, given $\kappa \in \mathcal{C}[\![\forall \underline{X}.\,\mathsf{B}]\!]_\gamma$, then, for any $\underline{A} \in \mathcal{A}$, we write $\kappa(\underline{A})$ for $\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_\gamma[j/\underline{X}])(\kappa_{\underline{C}})$, where $j\colon \underline{C} \multimap \underline{A}$ is an isomorphism and $\underline{C} \in \mathbf{A}$. The above notation defines the required projections exhibiting π and κ as elements of large products indexed by the objects of $\mathcal{C}$ and $\mathcal{A}$ respectively. The lemma below shows that the tuples π and κ remain parametric when considered as elements of the large products, i.e., that the derived projections preserve relations.[4]

Lemma 4.9.

(1) *If* $\mathcal{R}[\![\forall X.\,\mathsf{B}]\!]_\rho(\pi,\pi')$ *then, for all sets* A, C *in* $\mathcal{C}$ *and relations* $R \in \mathcal{R}_{\mathcal{C}}(A,C)$*, it holds that* $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}(\pi(A),\pi'(C))$.

(2) *If* $\mathcal{R}[\![\forall \underline{X}.\,\mathsf{B}]\!]_\rho(\kappa,\kappa')$ *then, for all algebras* $\underline{A},\underline{C}$ *in* $\mathcal{A}$ *and relations* $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{C})$*, it holds that* $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[Q/\underline{X}]}(\kappa(\underline{A}),\kappa'(\underline{C}))$.

Proof. We just prove item 1 of the lemma, item 2 is proved similarly. Suppose we are given sets A, C in $\mathcal{C}$ and a relation $R \in \mathcal{R}_{\mathcal{C}}(A,C)$. Then we know that there exists sets $A', C' \in \mathbf{C}$ and isomorphisms $i\colon A' \to A$, $j\colon C' \to C$. By definition, if $\mathcal{R}[\![\forall X.\,\mathsf{B}]\!]_\rho(\pi,\pi')$ then $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]}(\pi_{A'},\pi'_{C'})$ and so by Corollary 4.7

$$(\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i/X]), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j/X]))^{-1}\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}(\pi_{A'},\pi'_{C'}).$$

So $(\pi(A),\pi'(C)) = (\mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_1}[i^{-1}/X])(\pi_{A'}), \mathrm{gpd}[\![\mathsf{B}]\!](\mathrm{id}_{\rho_2}[j^{-1}/X])(\pi'_{C'}))$ are in $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}$. □

Lemma 4.10. *For any computation type* $\underline{\mathsf{B}}(\Theta,X)$*, any* Θ *environment* γ *and any* A *in* $\mathcal{C}$ *the projection* $p_A\colon \mathcal{A}[\![\forall X.\,\underline{\mathsf{B}}]\!]_\gamma \to \mathcal{A}[\![\underline{\mathsf{B}}]\!]_{\gamma[A/X]}$ *mapping* κ *to* $\kappa(A)$ *is a homomorphism. Similarly for any* $\underline{\mathsf{B}}(\Theta,\underline{X})$ *and any* $\underline{A} \in \mathcal{A}$ *the projection* $p_{\underline{A}}\colon \mathcal{A}[\![\forall \underline{X}.\,\underline{\mathsf{B}}]\!]_\gamma \to \mathcal{A}[\![\underline{\mathsf{B}}]\!]_{\gamma[\underline{A}/\underline{X}]}$ *is a homomorphism.*

Proof. Note first that for A in $\mathbf{C}$, the projection p_A is a homomorphism since $\mathcal{A}[\![\forall X.\,\underline{\mathsf{B}}]\!]_\gamma$ is defined as a representative of an $\mathcal{A}$-subobject of a $\mathbf{C}$ indexed $\mathcal{A}$-product and p_A is the inclusion of the subobject followed by the projection. In general, $p_A(\kappa)$ is defined to be

$$\mathrm{gpd}[\![\underline{\mathsf{B}}]\!](\mathrm{id}_\gamma[i/X])(p_{A'}(\kappa))$$

for any $A' \in \mathbf{C}$, and isomorphism $i\colon A' \to A$. Since by Proposition 4.5 $\mathrm{gpd}[\![\underline{B}]\!](\mathrm{id}_\gamma[i/X])$ is a homomorphism, we see that p_A is a composition of homomorphisms and so itself a homomorphism. The second half of the lemma is proved similarly. □

[4] In the conference version of this paper [19], we saved space by using fictitious large products in the definition of the interpretation of polymorphic types. Here, by giving the honest definition, and deriving the required consequences, we are providing the missing technical justification for the use of large products in *op. cit.*

$$
\begin{aligned}
[\![x]\!]_\gamma &= \gamma(x)\\
[\![\lambda x\colon \mathsf{B}.\, t]\!]_\gamma = [\![\lambda^\circ x\colon \underline{\mathsf{A}}.\, t]\!]_\gamma &= (d\colon \mathcal{C}[\![\mathsf{B}]\!]_\gamma \mapsto [\![t]\!]_{\gamma[d/x]})\\
[\![s(t)]\!]_\gamma &= [\![s]\!]_\gamma([\![t]\!]_\gamma)\\
[\![\Lambda X.\, t]\!]_\gamma &= \{[\![t]\!]_{\gamma[A/X]}\}_{A\in\mathbf{C}}\\
[\![t[\colon \forall X.\, \mathsf{B}](\mathsf{A})]\!]_\gamma &= ([\![t]\!]_\gamma)(\mathcal{C}[\![\mathsf{A}]\!]_\gamma)\\
[\![\Lambda \underline{X}.\, t]\!]_\gamma &= \{[\![t]\!]_{\gamma[\underline{A}/\underline{X}]}\}_{\underline{A}\in\mathbf{A}}\\
[\![t[\colon \forall \underline{X}.\, \mathsf{B}](\underline{\mathsf{A}})]\!]_\gamma &= ([\![t]\!]_\gamma)(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma)
\end{aligned}
$$

Figure 4: Interpretation of Terms

Next, we define the interpretation of terms. Given a context Γ with all free type variables in Θ, a Θ-Γ-environment is a function defined on both the type variables in Θ and the term variables in Γ, such that the restriction of γ to Θ is a Θ-environment, and, for every type assigment $x\colon \mathsf{B}$ in Γ, it holds that $\gamma(x) \in \mathcal{C}[\![\mathsf{B}]\!]_\gamma$. A term $\Gamma \mid \Delta \vdash_\Theta t\colon \mathsf{B}$ (i.e., such that $\mathrm{ftv}(\Gamma, \Delta, t, \mathsf{B}) \subseteq \Theta$) is interpreted as an element $[\![t]\!]_\gamma \in \mathcal{C}[\![\mathsf{B}]\!]_\gamma$, relative to any Θ-(Γ, Δ)-environment γ. The definition of $[\![t]\!]_\gamma$ is given in Figure 4. In the two clauses that apply to $t(\underline{\mathsf{A}})$, we distinguish between the cases for t of type $\forall X.\, \mathsf{B}$ and $\forall \underline{X}.\, \mathsf{B}$. Note that the definition of $[\![s(t)]\!]_\gamma$ applies uniformly, whether s has type $\mathsf{B} \to \mathsf{C}$ or $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$.

Proposition 4.11. *If* $\Gamma \mid \Delta \vdash_\Theta t\colon \mathsf{B}$ *then:*

(1) *(Well-definedness) For any* Θ-(Γ, Δ)-*environment* γ*, the value* $[\![t]\!]_\gamma \in \mathcal{C}[\![\mathsf{B}]\!]_\gamma$ *is well defined.*

(2) *(Relational invariance) For any relational* Θ-*environment* ρ*, and* Θ-(Γ, Δ)-*environments* γ_1, γ_2 *extending* ρ_1, ρ_2 *respectively, define*

$$\mathcal{R}[\![\Gamma]\!]_\rho(\gamma_1, \gamma_2) \Leftrightarrow \forall x\colon \mathsf{A} \in (\Gamma, \Delta).\, \mathcal{R}[\![\mathsf{A}]\!]_\rho(\gamma_1(x), \gamma_2(x)).$$

Then $\mathcal{R}[\![\Gamma]\!]_\rho(\gamma_1, \gamma_2)$ *implies* $\mathcal{R}[\![\mathsf{B}]\!]_\rho([\![t]\!]_{\gamma_1}, [\![t]\!]_{\gamma_2})$.

If $\Gamma \mid x\colon \underline{\mathsf{A}} \vdash_\Theta t\colon \underline{\mathsf{B}}$ *then:*

(3) *(Homomorphism property) For any* Θ-Γ-*environment* γ*, the function* $d \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma \mapsto [\![t]\!]_{\gamma[d/x]}$ *is a homomorphism from* $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma$ *to* $\mathcal{A}[\![\underline{\mathsf{B}}]\!]_\gamma$.

Proof (sketch). The three statements of the proposition are proved simultaneously by structural induction on t. Most of the cases are standard and we just show a few.

We prove the homomorphism property in the case of application of a polymorphic term $t\colon \forall X.\, \underline{B}$ to a value type A. By definition

$$[\![t(\mathsf{A})]\!]_\gamma = p_{\mathcal{C}[\![\mathsf{A}]\!]_\gamma}([\![t]\!]_\gamma)$$

and so by the induction hypothesis and Lemma 4.10 $d \mapsto [\![t(\mathsf{A})]\!]_{\gamma[d/x]}$ is a composition of homomorphisms.

The homomorphism property in the case of function application $t(s)$ for $t\colon \underline{\mathsf{B}} \multimap \underline{\mathsf{C}}$ follows from well definedness: by induction hypothesis $[\![t]\!]_\gamma \in \mathcal{C}[\![\underline{\mathsf{B}} \multimap \underline{\mathsf{C}}]\!]_\gamma$ and so is a homomorphism, so if $d \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma \mapsto [\![s]\!]_{\gamma[d/x]}$ is a homomorphism so is $d \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma \mapsto [\![t]\!]_\gamma([\![s]\!]_{\gamma[d/x]})$. Likewise well definedness in the case of linear lambda abstraction: $\lambda^\circ x\colon \underline{\mathsf{A}}.\, t$ follows from the homomorphism property for t.

We show well definedness in one of the cases of polymorphic lambda abstraction: $\Lambda X.\, t\colon \forall X.\, \mathsf{B}$. Here we must show that $\{[\![t]\!]_{\gamma[A/X]}\}_{A\in\mathbf{C}}$ satisfies the parametricity condition

in the definition of $\mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma$: for all $A, B \in \mathbf{C}$ and all relations $R \in \mathcal{R}_\mathcal{C}(A, B)$,

$$\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[R/X]}([\![t]\!]_{\gamma[A/X]}, [\![t]\!]_{\gamma[B/X]})$$

This follows from the relational invariance property for t, as assumed in the induction hypothesis, since $\mathcal{R}[\![\Gamma]\!]_{\Delta_\gamma}(\gamma, \gamma)$ holds by the identity extension lemma. Likewise, the relational invariance property in the case of type application of polymorphic terms follows from well definedness using Lemma 4.9.

To show relational invariance in case of polymorphic application at computation types $t(\underline{\mathsf{A}})$ we may use the induction hypothesis

$$\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\rho([\![t]\!]_{\gamma_1}, [\![t]\!]_{\gamma_2}).$$

From Lemma 4.9 it follows that

$$\mathcal{R}[\![\mathsf{B}]\!]_{\rho[\mathcal{R}[\![\underline{\mathsf{A}}]\!]\rho/\underline{X}]}([\![t]\!]_{\gamma_1}(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma_1}), [\![t]\!]_{\gamma_2}(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma_2})).$$

Finally, Lemma 4.2 implies

$$\mathcal{R}[\![\mathsf{B}[\underline{\mathsf{A}}/\underline{X}]]\!]_\rho([\![t(\underline{\mathsf{A}})]\!]_{\gamma_1}, [\![t(\underline{\mathsf{A}})]\!]_{\gamma_2})$$

as desired. □

Our main application of the model will be to establish semantic equalities between terms. Henceforth, for $\Gamma \mid \Delta \vdash s\colon \mathsf{B}$ and $\Gamma \mid \Delta \vdash t\colon \mathsf{B}$, we write $\Gamma \mid \Delta \vdash s = t\colon \mathsf{B}$ to mean that $[\![s]\!]_\gamma = [\![t]\!]_\gamma$ for all appropriate γ. For a syntactic equality theory we refer to [21].

5. Monadic types

In this section, we study the encoding of monadic types $!\mathsf{B}$ in our calculus, as defined by equation (1.1) of Section 1. One sees immediately that $!\mathsf{B}$ is always a computation type. We show that it enjoys the following derived introduction and elimination rules.

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{B}}{\Gamma \mid - \vdash !t\colon !\mathsf{B}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon !\mathsf{B} \quad \Gamma, x\colon \mathsf{B} \mid - \vdash u\colon \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash \text{let}\, !x\, \text{be}\, t\, \text{in}\, u\colon \underline{\mathsf{A}}}$$

Indeed, for this simply define:

$$!t =_{\text{def}} \Lambda \underline{X}.\, \lambda p\colon \mathsf{B} \to \underline{X}.\, p(t)$$
$$\text{let}\, !x\, \text{be}\, t\, \text{in}\, u =_{\text{def}} t(\underline{\mathsf{A}})(\lambda x\colon \mathsf{B}.\, u)\ .$$

It is the above rules that motivate our notation for the ! type constructor, since these are simply restrictions of the usual rules for the exponential ! of intuitionistic linear logic; for example, as formulated in Plotkin and Barber's DILL [2].

As a first application of relational parametricity for our system, we show that $!\mathsf{B}$ has the correct universal property for Moggi's monadic type. To keep the semantic notation bearable, we frequently omit semantic brackets, treating syntactic objects as the semantic elements they define, and we freely mix syntactic expressions with semantic values. For example, given any set A in $\mathcal{C}$, we simply write $!A$ rather than $\mathcal{C}[\![!\, X]\!]_{[A/X]}$ or $\mathcal{A}[\![!\, X]\!]_{[A/X]}$, referring to $!A$ as a set or as an algebra respectively when disambiguation is needed.

Lemma 5.1.

(1) *If* $\Gamma \mid - \vdash t\colon \mathsf{B}$ *and* $\Gamma, x\colon \mathsf{B} \mid - \vdash u\colon \underline{\mathsf{A}}$ *then* $\Gamma \mid - \vdash \text{let}\, !x\, \text{be}\, !t\, \text{in}\, u = u[t/x]\colon \underline{\mathsf{A}}$.
(2) $\Gamma \mid y\colon !\mathsf{A} \vdash y = \text{let}\, !x\, \text{be}\, y\, \text{in}\, !x\colon !\mathsf{A}$.

(3) *Suppose that* $\Gamma \mid \Delta \vdash s \colon !\,\mathsf{A}, \quad \Gamma, x \colon \mathsf{A} \mid - \vdash t \colon \underline{\mathsf{B}}$ *and* $\Gamma \mid y \colon \underline{\mathsf{B}} \vdash u \colon \underline{\mathsf{C}}$*, then* $\Gamma \mid \Delta \vdash \mathrm{let}\, !x \,\mathrm{be}\, s \,\mathrm{in}\, u[t/y] \;=\; u[\,\mathrm{let}\, !x \,\mathrm{be}\, s \,\mathrm{in}\, t \,/\, y] \colon \underline{\mathsf{C}}$.

Proof. Item 1 is a straightforward consequence of the semantic validity of beta equality.

For 2, we must show that $y = y(!\,\mathsf{A})(\lambda x \colon \mathsf{A}.\, !\,x)$ at type $\forall \underline{X}.\, (\mathsf{A} \to \underline{X}) \to \underline{X}$. By evident extensionality properties of the model, it suffices to show that, for any algebra $\underline{B}$ and $f \colon \mathsf{A} \to U\underline{B}$ in $\mathcal{C}$, we have $y(\underline{B})(f) = y(!\,\mathsf{A})(\lambda x \colon \mathsf{A}.\, !\,x)(\underline{B})(f)$.

Consider the homomorphism $g \colon !\,\mathsf{A} \multimap \underline{B}$ defined by $g(z) = z(\underline{B})(f)$. Then $\langle g \rangle$ is in $\mathcal{R}_{\mathcal{A}}(!\,\mathsf{A}, \underline{B})$. So, by parametricity,

$$((\Delta_{\mathsf{A}} \to \langle g \rangle) \to \langle g \rangle)\ (y(!\,\mathsf{A}),\, y(\underline{B}))\ . \tag{5.1}$$

For any $x \in \mathsf{A}$, we have $g(!x) = (\Lambda \underline{X}.\, \lambda p.\, p(x))(\underline{B})(f) = f(x)$, i.e.,

$$(\Delta_{\mathsf{A}} \to \langle g \rangle)\ (\lambda x \colon \mathsf{A}.\, !x,\, f)\ . \tag{5.2}$$

Combining (5.1) and (5.2), we obtain that

$$\langle g \rangle\ (y(!\,\mathsf{A})(\lambda x \colon \mathsf{A}.\, !x),\, y(\underline{B})(f))\ ,$$

i.e., $g(y(!\,\mathsf{A})(\lambda x \colon \mathsf{A}.\, !x)) = y(\underline{B})(f)$. Thus it indeed holds that

$$y(!\,\mathsf{A})(\lambda x \colon \mathsf{A}.\, !\,x)(\underline{B})(f) = y(\underline{B})(f)\,.$$

For 3, $h = \lambda^{\circ} y \colon \underline{\mathsf{B}}.\, u \colon \underline{\mathsf{B}} \multimap \underline{\mathsf{C}}$ is a homomorphism, so $\langle h \rangle \in \mathcal{R}_{\mathcal{A}}(\underline{\mathsf{B}}, \underline{\mathsf{C}})$. By parametricity, we have that

$$((\Delta_{\mathsf{A}} \to \langle h \rangle) \to \langle h \rangle)\ (s(\underline{\mathsf{B}}),\, s(\underline{\mathsf{C}}))\ . \tag{5.3}$$

Consider $\lambda x \colon \mathsf{A}.\, t \colon \mathsf{A} \to \underline{\mathsf{B}}$ and $\lambda x \colon \mathsf{A}.\, u[t/y] \colon \mathsf{A} \to \underline{\mathsf{C}}$. Then, for $x \in \mathsf{A}$, it holds that $h((\lambda x \colon \mathsf{A}.\, t)(x)) = u[t/y] = (\lambda x \colon \mathsf{A}.\, u[t/y])(x)$, i.e.,

$$(\Delta_{\mathsf{A}} \to \langle h \rangle)\ (\lambda x \colon \mathsf{A}.\, t,\, \lambda x \colon \mathsf{A}.\, u[t/y])\ . \tag{5.4}$$

Combining (5.3) and (5.4), we obtain that

$$\langle h \rangle\ (s(\underline{\mathsf{B}})(\lambda x \colon \mathsf{A}.\, t),\, s(\underline{\mathsf{C}})(\lambda x \colon \mathsf{A}.\, u[t/y]))\ ,$$

i.e., $h(s(\underline{\mathsf{B}})(\lambda x \colon \mathsf{A}.\, t)) = s(\underline{\mathsf{C}})(\lambda x \colon \mathsf{A}.\, u[t/y])$. So indeed we have $u[\,\mathrm{let}\, !x \,\mathrm{be}\, s \;\mathrm{in}\, t \,/\, y] = h(s(\underline{\mathsf{B}})(\lambda x \colon \mathsf{A}.\, t)) = s(\underline{\mathsf{C}})(\lambda x \colon \mathsf{A}.\, u[t/y]) = \mathrm{let}\, !x \,\mathrm{be}\, s \,\mathrm{in}\, u[t/y]$. □

Lemma 5.1 can be formulated as the two equality rules for the monadic type let constructor.

$$\frac{\Gamma \mid - \vdash t \colon \mathsf{B} \quad \Gamma,\, x \colon \mathsf{B} \mid - \vdash u \colon \underline{\mathsf{A}}}{\Gamma \mid - \vdash \mathrm{let}\, !\,x \,\mathrm{be}\, !\,t \,\mathrm{in}\, u \;=\; u[t/x] \colon \underline{\mathsf{A}}} \qquad \frac{\Gamma \mid \Delta \vdash s \colon !\,\mathsf{A} \quad \Gamma \mid y \colon !\,\mathsf{A} \vdash u \colon \underline{\mathsf{C}}}{\Gamma \mid \Delta \vdash \mathrm{let}\, !x \,\mathrm{be}\, s \,\mathrm{in}\, u[!\,x/y] \;=\; u[s \,/\, y] \colon \underline{\mathsf{C}}}$$

It is not hard to show that the two rules above are equivalent to the three items of Lemma 5.1 and we leave this as a straightforward exercise.

For any set A in $\mathcal{C}$ define $\eta_A \colon A \to !A$ by $\eta_A = \lambda x.\, !x$.

Theorem 5.2. *The function* $\eta_A \colon A \to !A$ *presents* $!A$ *as the free algebra over* A*, i.e., for any algebra* $\underline{B}$ *and function* $f \colon A \to U\underline{B}$*, there exists a unique homomorphism* $h \colon !A \multimap \underline{B}$ *such that* $h \circ \eta_A = f$*. Indeed,* h *is given by* $\lambda^{\circ} y.\, \mathrm{let}\, !x \,\mathrm{be}\, y \,\mathrm{in}\, f(x)$.

Proof. Clearly $\lambda^\circ y.\mathrm{let}\,!x\,\mathrm{be}\,y\,\mathrm{in}\,f(x)$ is a homomorphism, and $(\lambda^\circ y.\mathrm{let}\,!x\,\mathrm{be}\,y\,\mathrm{in}\,f(x))\circ\eta_A = f$ because $\mathrm{let}\,!x\,\mathrm{be}\,!x\,\mathrm{in}\,f(x) = f(x)$ by Lemma 5.1.1. For uniqueness, suppose h is such that $h\circ\eta_A = f$. Then

$$\begin{aligned} h(y) &= h(\mathrm{let}\,!\,x\,\mathrm{be}\,y\,\mathrm{in}\,!x) && \text{(Lemma 5.1.2)}\\ &= \mathrm{let}\,!\,x\,\mathrm{be}\,y\,\mathrm{in}\,h(!x) && \text{(Lemma 5.1.3)}\\ &= \mathrm{let}\,!\,x\,\mathrm{be}\,y\,\mathrm{in}\,f(x) && (h\circ\eta_A = f)\ , \end{aligned}$$

as required. □

It follows from the above theorem that the operation mapping A to the algebra $!A$ is the object part of a functor $F\colon \mathcal{C}\to\mathcal{A}$ left adjoint to U. We write T for the associated monad UF on $\mathcal{C}$.

The bijective correspondence of Theorem 5.2 can be expressed in the type theory PE as an isomorphism of (value) types between $!\mathsf{A}\multimap\underline{\mathsf{B}}$ and $\mathsf{A}\to\underline{\mathsf{B}}$ given by terms

$$\lambda f\colon \mathsf{A}\to\underline{\mathsf{B}}.\ \lambda^\circ z\colon !\mathsf{A}.\ \mathrm{let}\,!x\,\mathrm{be}\,z\,\mathrm{in}\,f(x)\colon (\mathsf{A}\to\underline{\mathsf{B}})\to !\mathsf{A}\multimap\underline{\mathsf{B}}$$
$$\lambda g\colon !\mathsf{A}\multimap\underline{\mathsf{B}}.\ \lambda x\colon \mathsf{A}.\ g(\eta_{\mathsf{A}}(x))\colon (!\mathsf{A}\multimap\underline{\mathsf{B}})\to\mathsf{A}\to\underline{\mathsf{B}}\ .$$

Thus we have a Girard decomposition of function spaces with computation type codomains, further motivating the ! notation.

We end this section with three characterisations of the induced relational lifting of the ! type constructor.

Proposition 5.3. *Suppose A, B are objects of $\mathcal{C}$ and $R\in\mathcal{R}_{\mathcal{C}}(A,B)$ is a relation.*

1. *$!\,R\in\mathcal{R}_{\mathcal{A}}(!\,A,!\,B)$ is the smallest admissible $\mathcal{A}$-relation containing all pairs of the form $(\eta(x),\eta(y))$ for $(x,y)\in R$.*
2. *$!\,R$ is the smallest admissible relation containing the image of the map $TR\to TA\times TB$ obtained by applying the functor T to the span corresponding to R.*
3. *If $\underline{A},\underline{B}\in\mathcal{A}$, $R\in\mathcal{R}_{\mathcal{C}}(A,B)$, $Q\in\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B})$ and $f\colon !\,A\multimap\underline{A}, g\colon !\,B\multimap\underline{B}$, then $(!\,R\multimap Q)(f,g)$ iff $(R\to Q)(f\circ\eta_A, g\circ\eta_B)$.*

Proof. For item 1 we first show that if $(x,y)\in R$ then $(\eta_A(x),\eta_B(y))\in !\,R$. So suppose we are given $\underline{A},\underline{B}\in\mathcal{A}$ and $Q\in\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B})$. We must show that if $f\colon A\to U\underline{A}, g\colon B\to U\underline{B}$ satisfy $(R\to Q)(f,g)$ then $Q(\eta_A(x)(\underline{A})(f),\eta_B(y)(\underline{B})(g))$. But this follows from definition of $(R\to Q)$ since $(\eta_A(x)(\underline{A})(f),\eta_B(y)(\underline{B})(g) = (f(x),g(y))$.

Now, suppose $Q\in\mathcal{R}_{\mathcal{A}}(!\,A,!\,B)$ and for all $(x,y)\in R$ we have $Q(\eta_A(x),\eta_B(y))$, or in other words $(R\to Q)(\eta_A,\eta_B)$. We must show that $!\,R\subseteq Q$. So suppose $!\,R(z,z')$. By definition of $!\,R$ using $(R\to Q)(\eta_A,\eta_B)$ we have

$$Q(z(!\,A)(\eta_A), z'(!\,B)(\eta_B)).$$

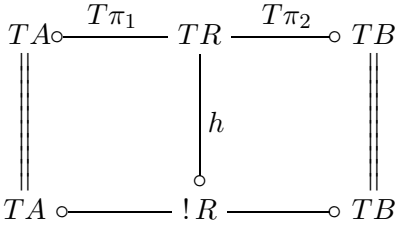
But by definition $z(!\,A)(\eta_A) = \mathrm{let}\,!\,x\,\mathrm{be}\,z\,\mathrm{in}\,!\,x$ which by Lemma 5.1 is equal to z. Likewise $z'(!\,B)(\eta_B) = z'$ proving $Q(z,z')$.

For the proof of item 2 we use the notation $\mathrm{im}(TR)^\circ$ for the smallest admissible relation containing the image of the map obtained by applying T to the span corresponding to R. Since $(R\to\mathrm{im}(TR)^\circ)(\eta_A,\eta_B)$, by item 1 the relation $!\,R$ is contained in $\mathrm{im}(TR)^\circ$. For the other inclusion notice that since $(R\to !\,R)(\eta_A,\eta_B)$, naturality of the correspondence given

$$
\begin{array}{rll}
1^\circ =_{\mathrm{def}} \forall \underline{X}.\, 0 \to \underline{X} & & \\
\underline{\mathsf{A}} \times^\circ \underline{\mathsf{B}} =_{\mathrm{def}} \forall \underline{X}.\, ((\underline{\mathsf{A}} \multimap \underline{X}) + (\underline{\mathsf{B}} \multimap \underline{X})) \to \underline{X} & & (\underline{X} \notin \mathrm{ftv}(\underline{\mathsf{A}},\underline{\mathsf{B}})) \\
0^\circ =_{\mathrm{def}} \forall \underline{X}.\, \underline{X} & & \\
\underline{\mathsf{A}} \oplus \underline{\mathsf{B}} =_{\mathrm{def}} \forall \underline{X}.\, (\underline{\mathsf{A}} \multimap \underline{X}) \to (\underline{\mathsf{B}} \multimap \underline{X}) \to \underline{X} & & (\underline{X} \notin \mathrm{ftv}(\underline{\mathsf{A}},\underline{\mathsf{B}})) \\
\mathsf{B} \cdot \underline{\mathsf{A}} =_{\mathrm{def}} \forall \underline{X}.\, (\mathsf{B} \to \underline{\mathsf{A}} \multimap \underline{X}) \to \underline{X} & & (\underline{X} \notin \mathrm{ftv}(\mathsf{B},\underline{\mathsf{A}})) \\
\exists^\circ X.\, \underline{\mathsf{A}} =_{\mathrm{def}} \forall \underline{Y}.\, (\forall X.\, (\underline{\mathsf{A}} \multimap \underline{Y})) \to \underline{Y} & & (\underline{Y} \notin \mathrm{ftv}(\underline{\mathsf{A}})) \\
\exists^\circ \underline{X}.\, \underline{\mathsf{A}} =_{\mathrm{def}} \forall \underline{Y}.\, (\forall \underline{X}.\, (\underline{\mathsf{A}} \multimap \underline{Y})) \to \underline{Y} & & (\underline{Y} \notin \mathrm{ftv}(\underline{\mathsf{A}})) \\
\mu^\circ \underline{X}.\, \underline{\mathsf{A}} =_{\mathrm{def}} \forall \underline{X}.\, (\underline{\mathsf{A}} \multimap \underline{X}) \to \underline{X} & & (\underline{X} \text{ +ve in } \underline{\mathsf{A}}) \\
\nu^\circ \underline{X}.\, \underline{\mathsf{A}} =_{\mathrm{def}} \exists^\circ \underline{X}.\, (\underline{X} \multimap \underline{\mathsf{A}}) \cdot \underline{X} & & (\underline{X} \text{ +ve in } \underline{\mathsf{A}})
\end{array}
$$

Figure 5: Definable computation types

by Theorem 5.2 implies the existence of a map h making the diagram



commute. This proves $\mathrm{im}(TR) \subseteq !\,R$. Since $!\,R$ is admissible $\mathrm{im}(TR)^\circ$ must be contained in $!\,R$.

For item 3 the "only if" direction is simply because $(R \to !\,R)(\eta_A, \eta_B)$. On the other hand, if $(R \to Q)(f \circ \eta_A, g \circ \eta_B)$ then $(f,g)^{-1}Q$ is an admissible relation containing all elements of the form $(\eta_A(x), \eta_B(y))$ for which $R(x,y)$ hold, and so by item 1 must contain $!\,R$ proving $(!\,R \multimap Q)(f,g)$. □

6. Definable computation types

The monadic type constructor ! is just one example of a type constructor definable using parametric polymorphism. In Figure 2 we have seen a collection of type constructors on value types and Figure 5 presents a collection of type constructors on computation types. The latter should be viewed as well chosen variants of Plotkin's polymorphic type encodings in second-order intuitionistic linear type theory, cf. [28, 3, 4]. (For relations between this calculus and PE see Section 8). We briefly discuss the computation type encodings.

Semantically, because $U\colon \mathcal{A} \to \mathcal{C}$ weakly creates limits, algebras are closed under products in $\mathcal{C}$. Syntactically, however, the types 1 and $\underline{\mathsf{A}} \times \underline{\mathsf{B}}$ from Figure 2 are *not* computation types. Thus the alternative encodings 1° and $\underline{\mathsf{A}} \times^\circ \underline{\mathsf{B}}$ are needed to obtain products of computation types as computation types. The types 0° and $\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}$ from Figure 5 define respectively an initial object and binary coproduct in the category $\mathcal{A}$. This structure in *not* preserved by U, and coproducts of algebras behave very differently from coproducts of sets in $\mathcal{C}$. (The latter are implemented by the sum types in Figure 2.) The type $\mathsf{B} \cdot \underline{\mathsf{A}}$ defines a $\mathcal{C}[\![\mathsf{B}]\!]$-fold copower of $\mathcal{A}[\![\underline{\mathsf{A}}]\!]$ in $\mathcal{A}$. Figure 5 also contains: existential types, $\exists^\circ X.\, \underline{\mathsf{A}}$ and $\exists^\circ \underline{X}.\, \underline{\mathsf{A}}$, packaged up as computation types; inductive computation types, $\mu^\circ \underline{X}.\, \underline{\mathsf{A}}$; and

coinductive computation types, $\nu^\circ \underline{X}.\, \underline{\mathsf{A}}$. As is standard, the (co)inductive types rely on the functoriality of type expressions in their positive arguments. A special case of the inductive types is the isomorphism

$$\underline{\mathsf{A}} \cong^\circ \forall \underline{X}.\, (\underline{\mathsf{A}} \multimap \underline{X}) \to \underline{X}$$

valid for all computation types $\underline{\mathsf{A}}$ in which $\underline{X}$ does not occur free. It is a consequence of relational parametricity that the above types all enjoy the correct universal properties. The arguments are carried out most naturally using a suitable logic for relational parametricity in PE, see [21].

7. Specialising the calculus to specific effects

The type theory PE is a generic calculus for effects since the type $!\mathsf{B}$ can be interpreted as an arbitrary monad, and no further effect-specific features are included. In this regard, PE is analogous to Moggi's computational λ-calculus [22], computational metalanguage [23] and Levy's call-by-push-value [15]. As with those calculi, specific effects can be incorporated by specialising the calculus appropriately. Typically, such specialisation takes place by extending the basic calculus with appropriately typed constants for any desired operations on effects. The addition of such constants takes place within the semantic theory described thus far, and so does not affect the validity of the results we have presented. For example, the universal properties of the defined types, discussed in Sections 5 and 6 (and treated in more detail in [21]), are unaltered.

In this section we consider various specialisations of the basic calculus, emphasising, in particular, the interaction with parametricity.

In a recent programme of research [31], Plotkin and Power have shown that many monads of computational interest can be profitably viewed as free algebra constructions for equational theories. This approach arises naturally from a computational viewpoint: the "algebraic operations" used to specify the theory correspond to programming primitives that cause effects, and the equational theory simply expresses natural behavioural equivalences between such primitives. We begin this section with an analysis of how to specialise PE to the case of such "algebraic effects".

Our approach is justified by a general theorem, which we now present. As one of their central results about algebraic effects, Plotkin and Power establish a one-to-one correspondence between "algebraic operations" and "generic effects" [30]. The theorem below reformulates this correspondence in our setting, and adds a third equivalent induced by our polymorphic description of monadic types. We shall apply this third equivalent to obtain the correct polymorphic typing for algebraic operations in effect-specific specialisations of PE.

Theorem 7.1. *For any set A in $\mathcal{C}$, there are one-to-one correspondences between:*

1. *"algebraic operations of arity A", i.e., natural transformations from the functor $(U(-))^A \colon \mathcal{A} \to \mathcal{C}$ to U,*
2. *"generic effects over A", i.e., elements of TA, and*
3. *"polymorphic computation type operations of arity A", that is, elements of the type $\forall \underline{X}.\, (A \to \underline{X}) \to \underline{X}$.*

The simplifications in the formulation of statement 1 above, compared with [30], are due to our set-theoretic setting, which renders it unnecessary to consider issues relating to

enrichment or tensorial strength. Also note that, by statement 2, the other two statements, in spite of appearances, depend only on the monad T on $\mathcal{C}$, not on how it is resolved into an adjunction $F \dashv U \colon \mathcal{A} \to \mathcal{C}$.

Proof. The equivalence of statements 2 and 3 is immediate from (1.1), because $TA = !A$. So we establish the equivalence of 1 and 3. Suppose that θ is a natural transformation from $(U(-))^A$ to U. We show that the mapping $\underline{A} \in \mathcal{A} \mapsto \lambda f \colon A \to U\underline{A}.\ \theta_{\underline{A}}(f)$ is an element of $\forall \underline{X}.\, (A \to \underline{X}) \to \underline{X}$. Suppose $\underline{A}, \underline{B} \in \mathcal{A}$ and $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B})$. We must show that if $(\Delta_A \to Q)\,(f, g)$ then also $Q(\theta_{\underline{A}}(f), \theta_{\underline{B}}(g))$. Since Q is an $\mathcal{A}$ relation there exists a span $\underline{A} \leftarrow \underline{C} \to \underline{B}$ in $\mathcal{A}$ projected by U to $U\underline{A} \leftarrow Q \to U\underline{B}$, and so by naturality the two squares below commute.

$$\begin{array}{ccccc}
(U\underline{A})^A & \xleftarrow{(\pi_1)^A} & Q^A & \xrightarrow{(\pi_2)^A} & U\underline{B}^A \\
\downarrow{\scriptstyle \theta_{\underline{A}}} & & \downarrow{\scriptstyle \theta_{\underline{C}}} & & \downarrow{\scriptstyle \theta_{\underline{B}}} \\
U\underline{A} & \xleftarrow[\pi_1]{} & Q & \xrightarrow[\pi_2]{} & U\underline{B}
\end{array}$$

But this says that, for any f, g with $Q(f(x), g(x))$ for all $x \in A$, it holds that $Q(\theta_{\underline{A}}(f), \theta_{\underline{B}}(g))$, which is what we needed to show. For the converse direction, suppose κ is an element of $\forall \underline{X}.\, (A \to \underline{X}) \to \underline{X}$. Then $\theta_{\underline{A}}(f) = \kappa(\underline{A})(f)$ is the corresponding algebraic operation. Verifying naturality is a routine use of graphs of homorphisms: if $g \colon \underline{B} \multimap \underline{C}$ and $f \colon A \to \underline{B}$ then by parametricity

$$((\Delta_A \to \langle g \rangle) \to \langle g \rangle)(\kappa(\underline{B}), \kappa(\underline{C}))\ ,$$

so since $(\Delta_A \to \langle g \rangle)(f, g \circ f)$, also $\langle g \rangle(\kappa(\underline{B})(f), \kappa(\underline{C})(g \circ f))$, i.e., $g(\theta_{\underline{B}}(f)) = \theta_{\underline{C}}(g \circ f)$ proving naturality. It is obvious that the two constructions are mutually inverse. □

To illustrate how Theorem 7.1 informs the specialisation of PE to algebraic effects, we consider nondeterminism as a typical example. As in [31], nondeterministic choice is naturally formulated using a binary operation "or" satisfying the semilattice equations:

$$x \,\mathrm{or}\, x = x, \quad x \,\mathrm{or}\, y = y \,\mathrm{or}\, x, \quad x \,\mathrm{or}\, (y \,\mathrm{or}\, z) = (x \,\mathrm{or}\, y) \,\mathrm{or}\, z\ .$$

Define the category $\mathcal{A}_{\mathrm{nd}}$ of "nondeterministic algebras" to have, as objects, structures (A, or_A) where A is a set in $\mathcal{C}$ and $\mathrm{or}_A \colon A \times A \to A$ satisfies the semilattice equations, and, as morphisms from (A, or_A) to (B, or_B), functions from A to B that are homomorphisms with respect to the "or" operations. It is easily verified that the obvious forgetful functor $U \colon \mathcal{A}_{\mathrm{nd}} \to \mathcal{C}$ satisfies conditions (A1)–(A4).

Since the morphisms in $\mathcal{A}_{\mathrm{nd}}$ are homomorphisms, the operation mapping any nondeterministic algebra (A, or_A) to the function $\mathrm{or}_A \colon A^2 \to A$ is an algebraic operation of arity 2 in the sense of statement 1 of Theorem 7.1. Thus, applying Theorem 7.1 and currying, one obtains a corresponding polymorphic operation:

$$\mathrm{or} : \ \forall \underline{X}.\ \underline{X} \to \underline{X} \to \underline{X}\ .$$

Accordingly, nondeterministic choice can be incorporated in PE by adding a constant "or", typed as above, to the type theory. This example illustrates the general pattern for adding algebraic operations as polymorphic constants to our type theory, and readily adapts to the algebraic operations associated with other algebraic effects.

A limitation of the notion of algebraic operation is that there exist effect-specific programming primitives that are not algebraic operations. One well-known example of such a primitive is exception handling. Below, we show how exception handling may also be incorporated within our approach as a suitably typed polymorphic constant. The approach is justified by a general theorem, giving another instance of a coincidence between natural transformations and elements of polymorphic type.

Theorem 7.2. *For any $n \in \mathbb{N}$, there are one-to-one correspondences between:*
(1) *Natural transformations from $(F(-))^n \colon \mathcal{C} \to \mathcal{A}$ to $F \colon \mathcal{C} \to \mathcal{A}$, and*
(2) *elements of $\forall X.\, (n \to !X) \multimap !X$,*
where, in statement 2, we write n for the n-fold coproduct type $1 + \cdots + 1$, as defined in Figure 2.

Proof. An element of $\forall X.\, (n \to !\, X) \multimap !\, X$ gives for each $A \in \mathcal{C}$ a map $(FA)^n \multimap FA$, and the naturality square for this family follows from the parametricity condition satisfied by elements of polymorphic type, applied to the graph of a function. The interesting part of this proof is to show that natural transformations satisfy the parametricity condition and thus define elements of $\forall X.\, (n \to !\, X) \multimap !\, X$.

So suppose $(f_A \colon (FA)^n \multimap FA)_{A \in \mathcal{C}}$ is a natural transformation, and $A, B \in \mathcal{C}$ and $R \in \mathcal{R}_{\mathcal{C}}(A, B)$. We must show that $((!\, R)^n \multimap !\, R)(f_A, f_B)$. Naturality applied to the span $A \leftarrow R \rightarrow B$ gives us commutativity of

$$\begin{array}{ccccc} (FA)^n & \xleftarrow{(F\pi_1)^n} & (FR)^n & \xrightarrow{(F\pi_2)^n} & (FB)^n \\ \downarrow{\scriptstyle f_B} & & \downarrow{\scriptstyle f_R} & & \downarrow{\scriptstyle f_B} \\ FA & \xleftarrow{F\pi_1} & FR & \xrightarrow{F\pi_2} & FB \end{array}$$

Since f_A and f_B are homomorphisms, this implies

$$(\mathrm{im}((TR)^n)^\circ \multimap \mathrm{im}(TR)^\circ)(Uf_A, Uf_B)$$

Now, one can easily check that $\mathrm{im}((TR)^n)^\circ = (\mathrm{im}(TR)^\circ)^n$ and so $((!\, R)^n \multimap !\, R)(Uf_A, Uf_B)$ by Proposition 5.3, as desired. □

We now consider exception handling in detail. We assume we have a set E of exceptions with decidable equality (i.e., for all $e, e' \in E$ either $e = e'$ or $e \neq e'$). We also assume (for simplicity) that $\mathcal{C}$ is closed under binary coproduct in **Set** (this is consistent with the axioms for $\mathcal{C}$). We define the category $\mathcal{A}_{\mathrm{exc}}$ of "exception algebras" to have, as objects, structures $(A, \{\mathrm{raise}^e_A\}_{e \in E})$ where $\mathrm{raise}^e_A \in A$, and, as morphisms from $(A, \{\mathrm{raise}^e_A\}_{e \in E})$ to $(B, \{\mathrm{raise}^e_B\}_{e \in E})$, functions from A to B that map each raise^e_A to raise^e_B. Since the raise^e elements are algebraic constants (operations of arity 0), they can be added to PE as constants:

$$\mathrm{raise}^e \colon \forall \underline{X}.\, \underline{X} \ .$$

As is standard, the forgetful functor from $\mathcal{A}_{\mathrm{exc}}$ to $\mathcal{C}$, has as its left adjoint the functor F mapping A to the exception algebra $(A + E, \{\mathrm{inr}(e)\}_{e \in E})$. For an exception $e \in E$, the handling operation over A is the function $\mathrm{handle}^e_A \colon (F(A))^2 \to F(A)$ defined by

$$\mathrm{handle}^e_A(p, q) \ = \ \begin{cases} p & \text{if } p \neq \mathrm{inr}(e) \\ q & \text{if } p = \mathrm{inr}(e) \ . \end{cases}$$

It is easily shown that this specifies a natural transformation from the functor $(F(-))^2 \colon \mathcal{C} \to \mathcal{A}_{\mathrm{exc}}$ to $F \colon \mathcal{C} \to \mathcal{A}_{\mathrm{exc}}$. In particular, the component handle^e_A of the natural transformation does lie in $\mathcal{A}_{\mathrm{exc}}$ because the interpretation of raise^e in the exception algebra $F(A)^2$ is the pair $(\mathrm{inr}(e), \mathrm{inr}(e))$. Thus, by Theorem 7.2, exception handling can be incorporated in PE by adding typed constants:

$$\mathrm{handle}^e \colon \forall X.\, (2 \to !X) \multimap !X \ .$$

The main surprise with this typing is that exception handling is given a "linear" type. From this typing, one of course obtains an associated term of the less informative type $\forall X.\, (2 \to !X) \to !X$, which is isomorphic to the expected type $\forall X.\, !X \to !X \to !X$.

Paul Levy (personal communication) has pointed out that the above account of exception handling is not robust, in the sense that, in the presence of effects other than exceptions, the linear typing of handle^e above is not always correct. In situations in which handling is non-linear, one would expect the non-linear typing $\forall X.\, !X \to !X \to !X$ to still be correct. However, Theorem 7.2 is no longer applicable to establish parametricity. It would thus be interesting to find a general argument, valid in the presence of other effects, for the parametricity of handling.

Both Theorems 7.1 and 7.2 relate elements of certain polymorphic types with natural transformations between associated functors. In fact, more generally, for types that determine functors, parametricity implies naturality (cf. [29]). However, the exact correspondences between natural transformations and parametric elements established above depend crucially on the precise forms of types considered there.

The forms of n-ary operation considered in this section by no means exhaust the collection of operations of interest from an effects perspective. Control operators provide a particularly interesting class of examples that do not fit into this format. We briefly discuss how PE can be specialised to control at the end of Section 8.

8. Relation to other systems

Several computational effects of interest, including nontermination, nondeterminism, and probabilistic choice, give rise to monads on $\mathcal{C}$ that are *commutative*, cf. [23]. The collection of models of PE in which $\mathcal{A}$ is the category of algebras for a commutative monad T is of special interest since, for such monads, the set of homomorphisms $\underline{A} \multimap \underline{B}$ between algebras $\underline{A}, \underline{B}$ carries a canonical algebra structure which provides a closed structure on the category $\mathcal{A}$. For such models, it is thus natural to modify our type system by including $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a computation type. Making this adjustment, one obtains second-order intuitionistic linear type theory as the fragment of computation types:

$$\underline{X} \mid \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \mid \underline{\mathsf{A}} \to \underline{\mathsf{B}} \mid \forall \underline{X}.\, \underline{\mathsf{A}} \ . \tag{8.1}$$

Thus we obtain a rich collection of models for the type theory proposed by Plotkin as a foundation for combining polymorphism and recursion [28].

A simple application of the polymorphic encodings in Figures 2 and 5 is to translate Levy's CBPV calculus [15] into PE. For this, coproducts and products of value types are translated using $+$ and $\times$ from Figure 2, products of computation types are translated using $\times^\circ$ from Figure 5, Levy's F constructor is translated using $!$, and U is simply ignored.

One of the properties of Levy's CBPV calculus is that its adjunction models [16] are not required to satisfy any properties analogous to our conditions (A1) and (A2). In Sections 3 and 4, we exploited (A1) to satisfy the requirement that $U(\mathcal{A}[\![\underline{\mathsf{A}}]\!]) = \mathcal{C}[\![\underline{\mathsf{A}}]\!]$, and (A2) to

obtain that relations in $\mathcal{A}$ can be viewed as special relations in $\mathcal{C}$ (cf. Lemma 3.2), which is crucial in interpreting $\mathcal{R}[\![\underline{\mathsf{A}}]\!]$ as an admissible $\mathcal{A}$-relation. We comment, however, that it is possible to generalise our account of relational parametricity to models in which (A1) is weakened to the requirement that $\mathcal{A}$ be small-complete and U preserve limits (which always holds in Levy's models since U is a right adjoint), and in which condition (A2) is dropped altogether. For such models, condition (A1) can then be engineered by changing $\mathcal{A}$ to an equivalent category, and adjusting U accordingly, as in [20]; or, more naturally, the semantics can be adjusted, rather than the category, so as to obtain a specified isomorphism $U(\mathcal{A}[\![\underline{\mathsf{A}}]\!]) \cong \mathcal{C}[\![\underline{\mathsf{A}}]\!]$, instead of an equality. Dropping condition (A2) causes a more significant complication. In its absence, it seems necessary to define a special relational semantics for computation types, rather than inheriting the relational semantics for computation types from that for value types (as done in Section 4). Moreover, while such an approach is natural, it does make the semantic definitions significantly more complicated. In this paper, we have chosen to assume properties (A1) and (A2), since we value the convenience of simplified semantic definitions (which are anyway complicated enough as they are!) over the added generality of having a wider class of models.

Finally, we mention how the interesting case of control operators can be accommodated within PE. This cannot be achieved by following the general methods of Section 7, since the continuations monad $R^{R^{(-)}}$ does not arise naturally as the free algebra for an algebraic theory, and the control primitives associated with continuations are not algebraic operations. Nevertheless, it turns out that PE can be usefully specialised to the case of control by adding a polymorphic constant of type (using the defined type 0° from Figure 5):

$$\forall \underline{X}.\, ((\underline{X} \multimap 0^\circ) \to 0^\circ) \;\multimap\; \underline{X} \ ,$$

acting as a pointwise inverse to the canonical element of type $\forall \underline{X}.\, \underline{X} \multimap ((\underline{X} \multimap 0^\circ) \to 0^\circ)$. The resulting theory is studied in detail in a companion article [20], where it is shown that Hasegawa's results on polymorphic definability in the second-order $\lambda\mu$-calculus [9] fall out as special cases of constructions from Figure 5.

9. Applicability of results

We have given a semantic account of relational parametricity in the presence of computational effects. From our working perspective within IZF, this is parametrized on being given categories $\mathcal{C}$ and $\mathcal{A}$ and families of relations $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{A}$, satisfying axioms (C1)–(C4), (A1)–(A4) and (R1)–(R4). Moreover, Proposition 3.5, shows that such data can be obtained whenever one has a monad T on a category $\mathcal{C}$ satisfying (C1)–(C4).

To conclude the paper, we outline how this theory might actually be applied to prove properties of polymorphic programs with effects. Suppose we have some given polymorphic λ-calculus $\mathbf{L}$ with a choice of effect-primitives as the programmming language of interest. The basic idea is to formulate both the operational and denotational semantics of $\mathbf{L}$ within IZF. The operational semantics is treated in the standard way, for which the use of classical logic is inessential. The denotational semantics is developed using the assumption of a category $\mathcal{C}$ satisfying (C1)–(C4). The construction of $\mathcal{A}$ and $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{A}$ will depend upon the effects present in the language. For (a simple) example, if the only effect is nondeterministic choice then T can be defined to be the free-semilattice functor over $\mathcal{C}$, and the entire model is then obtained via Proposition 3.5. For general effects, the construction of the model will be more complex than this, especially in the presence of recursion, cf. [35].

Indeed, there is need for a uniform theory of how to build such models; some hints in this direction appear in [40].

Once one has both operational semantics and model, the next step is to prove, within IZF, a *computational adequacy* result for the model, implying that the model is sound with respect to operational equivalence. In examples considered hitherto, such proofs have been obtained by standard logical-relations-based methods [37, 39, 35]. They rely only on having some appropriate non-triviality property of $\mathcal{C}$ (for example, that the natural numbers is an object of $\mathcal{C}$ [37]).

Computational adequacy allows one to transfer equational properties of the denotational semantics to the operational semantics. However, the above development has taken place in IZF, together with the assumption of a category $\mathcal{C}$ satisfying (C1)–(C4). We can therefore infer operational properties within this metatheory; but, of course, we want to be sure that such properties are actually true in the real world. The remaining step is to use a transfer property which allows us to conclude exactly this.

The transfer property is based on the existence of realizability models of IZF which possess within them categories $\mathcal{C}$ satisfying (C1)–(C4) and containing the natural numbers as an object. As already discussed in Section 3, such models derive from the work of Hyland *et. al.* on small-complete small categories [10, 12]. Now, the relevant realizability models all enjoy the property of being Π^0_2-*absolute*, meaning that a Π^0_2-sentence holds in the model if and only if it is true externally. This implies that properties of operational equivalence that are true in the model are indeed true in reality, see [37, 39, 35] for related arguments.

We have outlined a programme of how one can potentially use the theory of parametricity developed in this paper to derive operational properties of programs. It would be good to have examples of such applications worked out in computationally interesting cases.

There is, of course, a significant drawback with the intuitionistic-set-theory-based approach we have been following. The mathematical overheads are considerable. It seems likely that a more practical theory of parametricity for effects should be achievable using direct operational methods. We leave this as an interesting direction for future research. It is plausible that the denotational approach we have been following in this paper might be useful in informing the development of such an operational theory.

Acknowledgements

We are indebted to Masahito Hasegawa for first suggesting that the polymorphic definition of $!\mathsf{B}$ given by (1.1) should be a general phenomenon within a monad-based framework incorporating both linear and continuation-passing settings as special cases. We thank both him and Paul Levy for very helpful discussions, and the anonymous referees for useful suggestions.

References

[1] P. Aczel and M. Rathjen. Notes on Constructive Set Theory. Technical Report 40, Mittag-Leffler Institute, 2001.

[2] A. Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, University of Edinburgh, 1997.

[3] G. Bierman, A. Pitts, and C. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. *ENTCS*, 41:70–88, 2000.

[4] L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Linear Abadi & Plotkin logic. *Logical Methods in Computer Science*, 2, 2006.
[5] A. Filinski. *Controlling Effects*. PhD thesis, School of Computer Science, CMU, 1996.
[6] J.-Y. Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1: 255–296, 1991.
[7] J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.
[8] J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. *Mathematical Structures in Computer Science*, 18: 1169–1217, 2008.
[9] M. Hasegawa. Relational parametricity and control. *Logical Methods in Computer Science*, 2, 2006. Special issue for selected papers from LICS 2005.
[10] J.M.E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40:135 – 165, 1988.
[11] J.M.E. Hyland. First steps in synthetic domain theory. *Proc. of the 1990 Como Category Theory Conference*, pp. 131–156, Springer LNM 1488, 1991.
[12] J.M.E. Hyland, E. Robinson, and G. Rosolini. The discrete objects in the effective topos. *Proc. LMS.*, 3(60), 1990.
[13] A. Joyal and I. Moerdijk. *Algebraic Set Theory*. LMS Lecture Note Series 220, CUP, 1995.
[14] S. Katsumata. A semantic formulation of $\top\top$-lifting and logical predicates for computational metalanguage. In *Computer Science Logic*, Springer LNCS 3634, 2005.
[15] P.B. Levy. *Call-By-Push-Value*. Springer, 2004.
[16] P.B. Levy. Adjunction models for call-by-push-value with stacks. *Theory and Applications of Categories*, 14:75–110, 2005.
[17] S. Mac Lane. *Categories for the Working Mathematician*. Springer Graduate Texts in Mathematics, 1971.
[18] R.E. Møgelberg, L. Birkedal and G. Rosolini. Synthetic domain theory and models of linear Abadi & Plotkin logic. *Annals of Pure and Applied Logic*, 155:115–133, 2008.
[19] R.E. Møgelberg and A. Simpson. Relational Parametricity for Computational Effects. In *Proc. 22nd LICS Symposium*, pages 346–355, 2007.
[20] R.E. Møgelberg and A. Simpson. Relational Parametricity for Control Considered as a Computational Effect. In *Proc. MFPS XXIII, ENTCS* 173:295–312, 2007.
[21] R.E. Møgelberg and A. Simpson. A logic for parametric polymorphism with effects. In *TYPES*, volume 4941 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2007.
[22] E. Moggi. Computational lambda-calculus and monads. In *Proc. 4th LICS Symposium*, pages 14–23, 1989.
[23] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.
[24] M. Parigot. Strong normalization for second order classical natural deduction. *J. Symb. Logic*, 62:1461–1479, 1997.
[25] A.M. Pitts. Polymorphism is set theoretic, constructively. In *Proc. CTCS*, pages 12–39. Springer LNCS 283, 1987.
[26] A.M. Pitts. Non-trivial power types can't be subtypes of polymorphic types. In *Proc. 4th LICS Symposium*, pages 6–13, 1989.
[27] A.M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.
[28] G. Plotkin. Type theory and recursion (extended abstract). In *Proc. 8th LICS Symposium*, page 374, 1993.
[29] G. Plotkin and M. Abadi. A logic for parametric polymorphism. *Proc. TLCA*, pp.361–375. Springer LNCS 664, 1993.
[30] G. Plotkin and A.J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11:69–94, 2003.
[31] G. Plotkin and A.J. Power. Computational effects and operations: an overview. *ENTCS*, 73:149–163, 2004.
[32] J. Reynolds. Types, abstraction and parametric polymorphism. In *Inf. Processing*, pp.513–523. N. Holland, 1983.
[33] J. Reynolds. Polymorphism is not set-theoretic. In *Semantics of Data Types*. Springer LNCS 173, 1984.
[34] E. Robinson. How complete is PER? In *Proc. 4th LICS Symposium*, pages 106–111, 1989.

[35] G. Rosolini and A. Simpson. *Using Synthetic Domain Theory to Prove Operational Properties of a Polymorphic Programming Language Based on Strictness*. Preprint, 2004.
[36] A. Ščedrov. Intuitionistic set theory. In *Harvey Friedman's Research on The Foundations of Mathematics*, pages 257–284. Elsevier Science Publishers, 1985.
[37] A. Simpson. Computational adequacy in an elementary topos. In *Computer Science Logic*, Springer LNCS 1585, pp. 232–242, 1999.
[38] A. Simpson. Elementary axioms for categories of classes (extended abstract). In *Proc. 14th LICS Symposium*, pp. 77–85, 1999.
[39] A. Simpson. Computational adequacy for recursive types in models of intuitionistic set theory. *Annals of Pure and Applied Logic*, 130:207–275, 2004.
[40] A. Simpson. Beyond Classical Domain Theory. Tutorial given at MFPS XXIII, New Orleans, 2007.
[41] P. Wadler. Theorems for free! In *Proc. 4th Int. Conf. on Funct. Prog. Languages and Computer Arch.* London, 1989.